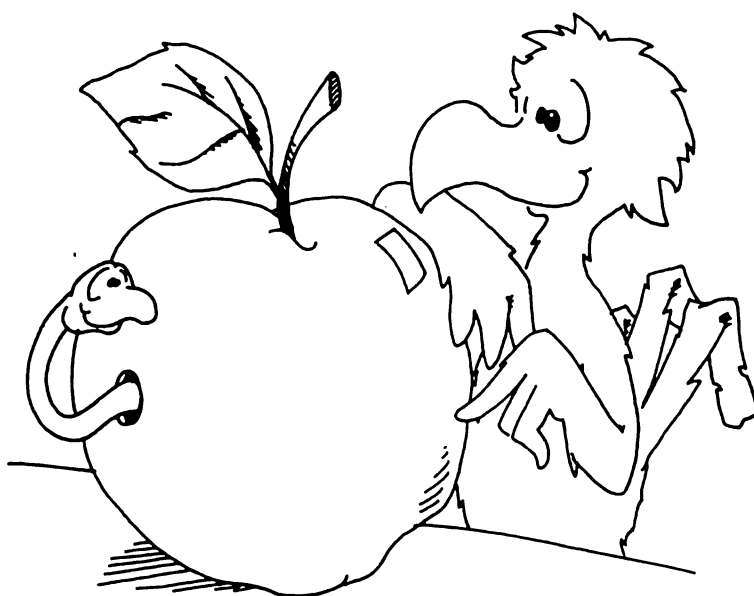






# KIDS AND THE APPLE



**BY**

**Edward H. Carlson**

**Department of Physics and Astronomy  
Michigan State University**

**Illustrated by**

**Paul D. Trap**

First Printing July 1982  
Second Printing January 1983  
Third Printing June 1983



8943 Fullbright Avenue  
Chatsworth, CA 91311-2750  
(213) 709-1202

ISBN No. 0-88190-019-2

**COPYRIGHT © 1982 BY DATAMOST**

This manual is published and copyrighted by DATAMOST. All rights are reserved by DATAMOST. Copying, duplicating, selling or otherwise distributing this product is hereby expressly forbidden except by prior written consent of DATAMOST.

The word APPLE and the Apple logo are registered trademarks of APPLE COMPUTER, INC.

APPLE COMPUTER, INC. was not in any way involved in the writing or other preparation of this manual, nor were the facts presented here reviewed for accuracy by that company. Use of the term APPLE should not be construed to represent any endorsement, official or otherwise, by APPLE COMPUTER, INC.



# TABLE OF CONTENTS

	Page
Acknowledgements .....	i
To The Kids .....	ii
To The Parents .....	iii
To The Teachers .....	iv
About Programming .....	v
About the Book .....	vi

## INTRODUCTION

1 Home, Print, New, and Run .....	7
2 Peeping, Flash, Inverse, and Normal .....	13
3 List and Rem .....	17
4 Special Keys .....	23
5 The INPUT Command .....	27
6 Tricks with Print, Speed .....	30
7 The LET Command .....	34
8 The GOTO Command and Reset Key .....	38
9 The IF Command .....	42
10 Introducing Numbers .....	48
11 Tab and Delay Loops .....	54
12 The IF Command with Numbers .....	59
13 Random Numbers and the INT Function .....	63
14 Save to the Disk .....	69

## GRAPHICS, GAMES, AND ALL THAT

15 Some Shortcuts .....	73
16 Moving About on the Screen, VTAB, HTAB .....	79
17 FOR-NEXT Loops .....	82
18 Edit and Run Modes, The Calculator .....	87
19 Moving Pictures Using Strings .....	91
20 Variable Names .....	95
21 LO-RES Graphics .....	98
22 Graphics Using HLIN and VLIN .....	103
23 Secret Writing and the GET Command .....	108
24 Pretty Programs, GOSUB, Return, End .....	113

## ADVANCED PROGRAMMING

25 Line Editing .....	119
26 Snipping Strings: LEFT\$, MID\$, RIGHT\$, LEN .....	123
27 Switching Numbers with Strings .....	128
28 Paddles for Action Games .....	132
29 ASCII Code, Keyboard, ON . . . GOTO .....	138
30 Arrays and the DIM Command .....	145
31 Logic: AND, OR, NOT .....	149
32 User Friendly Programs .....	156
33 Debugging, STOP, CTRL-C, CONT .....	162
Appendix A — Disk Usage .....	167
Appendix B — Saving to Tape .....	169
Appendix C — Reserved Words in Applesoft .....	172
Answers to Assignments .....	173
Glossary .....	207
Index of Commands .....	216
Error Messages .....	217
Index of Topics .....	219



## **ACKNOWLEDGEMENTS**

My sincere thanks go to Paul Sheldon Foote for suggesting I write this book. Paul pointed out to me that parents urgently need a book to help their children learn programming on their home computers. I have an Apple and kids at home and was helping plan and preparing to teach in the "Computers and You" summer camp at Michigan State University, but I had not noticed the absence of a book that would teach serious computing to children of middle school age.

I am deeply grateful to my fellow teachers and board members at the summer camp: Mark Lardie, John Forsyth, and Mary Winter, each of whom shared their teaching experiences with me and suggested techniques for communicating the material in an effective way.

Mark Lardie has been especially generous in reading the typescript and offering suggestions from his extensive experience in teaching computing to children under a variety of formats.

The summer camp experience was enhanced by its smooth operation under the direction of Marc Van Wormer and by the expert and sympathetic help of the student assistants Paul Johns and Cecilia Stauffer.

Remembering the enthusiastic pleasure of the summer camp students has encouraged me during the months spent in preparing this book. Several families have used the book in their homes and offered suggestions for improvement.

I especially wish to thank Steve Peter and his girls Karen and Kristy; George Campbell and his youngsters Andrew and Sarah; Beth O'Malia and Scott, John and Matt; Chris Clark and Chris Jr., Tryn, Daniel, and Vicky; and Paul Foote and David.

In addition, Lucy Slinger commented on the book from her perspective as a science teacher and Bill Brown from his extensive experience in microcomputing.

My own family has tolerated my periods of seclusion in the writing den. These were punctuated by noisy tours of the house in search of whichever of my children could listen to and evaluate my latest idea. So my final and heartfelt thanks go to my wife Louise and our children Karen, Brian and Minda.

## TO THE KIDS

This book teaches you how to write programs for the Apple computer.

You will learn how to make your own action games, board games and word games. You may entertain your friends with challenging games and provide some silly moments at your parties with short games you invent.

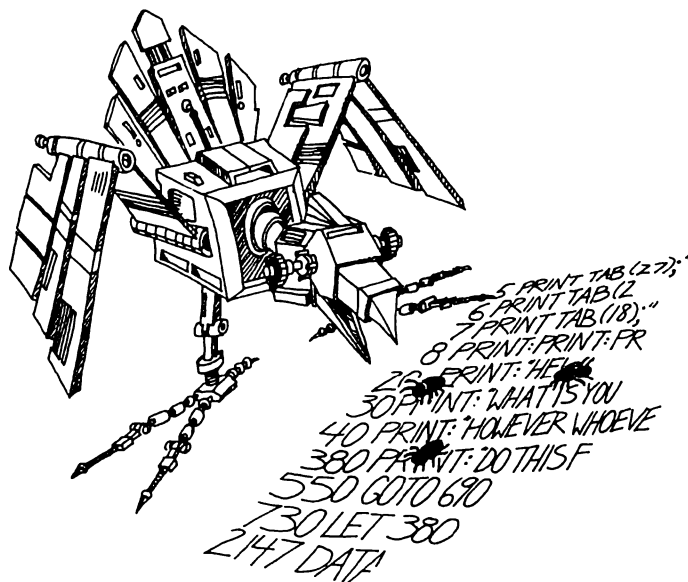
Perhaps your record collection or your paper route needs the organization your special programs can provide. If you are working on the school yearbook, maybe a program to handle the finances or records would be useful.

You may help your younger sisters and brothers by writing drill programs for arithmetic facts or spelling. Even your own schoolwork in history or foreign language may be made easier by programs you write.

**How to Use This Book** Do all the examples. Try all the assignments. If you get stuck, first go back and reread the lesson carefully from the top. You may have overlooked some detail. After trying hard to get unstuck by yourself, you may go ask a parent or teacher for help.

There are review questions for each lesson. Be sure you can answer them before announcing that you have finished the lesson!

MAY THE BLUEBIRD OF HAPPINESS EAT ALL THE BUGS IN YOUR PROGRAMS!





## TO THE PARENTS

This book is designed to teach Applesoft BASIC to youngsters in the range from 10 to 14 years old. It gives guidance, explanations, exercises, reviews, and “quizzes.” Some exercises have room for the student to write in answers that you can check later. Answers are provided in the back of the book for all assignments. Your child will probably need some help in getting started and a great deal of encouragement at the sticky places.

Learning to program is not easy because it requires handling some sophisticated concepts. It also requires accuracy and attention to detail which are not typical childhood traits. For these very reasons it is a valuable experience for children. They will be well rewarded if they can stick with the book long enough to reach the fun projects that are possible once a repertoire of commands is built up.

**How to Use the Book** The book is divided into 33 lessons for the kids to do. Each lesson is preceded by a NOTES section which you should read. It outlines the things to be studied, gives some helpful hints, and provides questions which you can use verbally (usually at the computer) to see if the skills and concepts have been mastered.

These notes are intended for the parents, but the older students may also profit by reading them. The younger students will probably not read them, and can get all the material they need from the lessons themselves. For the youngest children, it may be advisable to read the lesson out loud with them and discuss it before they start work.

## TO THE TEACHER

This book is designed for students in about the 7th grade. It teaches Applesoft BASIC on disk based or cassette Apple systems. The lessons contain explanations (including cartoons), examples, exercises, and review questions. Notes for the instructor which accompany each lesson summarize the material, provide helpful hints, and give good review questions.

The book is intended for self study, but may also be used in a classroom setting.

I view this book as teaching programming in the broadest sense, using the BASIC language, rather than teaching “BASIC.” Seymour Papert has pointed out in MINDSTORMS that programming can teach powerful ideas. Among these are the idea that procedures are entities in themselves. They can be named, broken down into elementary parts, and debugged. Some other concepts include these: “chunking” ideas into “mind sized bites,” organizing such modules in a hierarchical system, looping to repeat modules, and conditional testing (the IF . . . THEN statement).

Each concept is tied to everyday experiences of the student through choice of language to express the idea, through choice of examples, and through cartoons. Thus metaphor is utilized in making the “new” material familiar to the student.

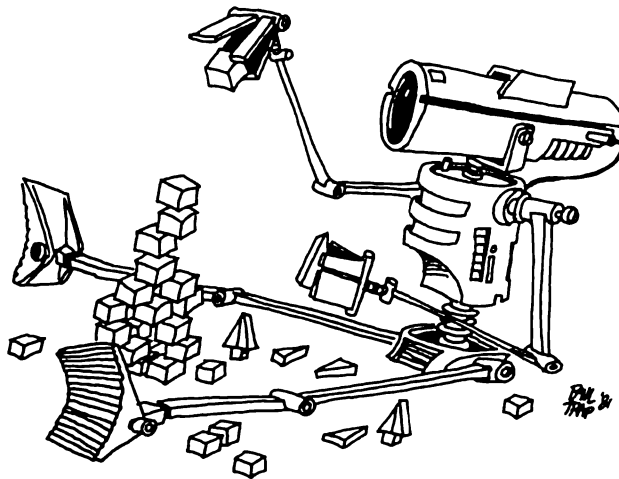
## ABOUT PROGRAMMING

There is a common misconception that programming a computer must be very similar to doing arithmetic. Not so. The childhood activities that computing most resembles may be playing with building blocks and writing an English composition.

Like a set of blocks, BASIC uses many copies of a small number of elements (commands) that are combined in rather standardized ways to achieve an original end result. As familiarity with the system grows, a “bag of tricks” is collected by the programmer that allows each command to serve a larger number of functions, just as the child first uses the “triangle block” in making roofs, but later finds that two of them make a splendid fir tree.

Like an essay, a program is a finished product that fulfills a specified need. The child writing to the theme “How I spent my summer,” adopts one of several working styles. The beginner may be hung up in how to hold the pencil and how to spell. The same child a few grades later will just start writing, not spending much time in forming good paragraphs, much less in planning the overall structure of the composition. With maturity comes freedom to move back and forth among the levels of concern, now thinking about the overall form, and a few moments later paying attention to word choice or punctuation.

Computing does have some similarities to arithmetic as seen by most children. There are rigid rules to learn: procedures in arithmetic but only syntax in programming. Even the tiniest mistake makes the whole result “wrong.” (A more effective attitude in programming is that “wrong” results are partly right, and need debugging, a normal and expected activity.) However, the limited scope for creativity in arithmetic contrasts sharply with the emphasis on creativity in program writing.



Programming offers general education advantages not easily found elsewhere in a child's experiences. The plasticity of the form, words on a screen that are created and destroyed by the touch of a key, allows effort to be concentrated on the central features to be learned. These features are balanced between analysis (why doesn't it work as I want) and synthesis (planning on several size scales, from the program as a whole down through loops and subroutines to individual commands). Learning on the computer is efficient of effort. Errors of syntax are automatically pointed out by the computer.

The analytical and synthetical skills learned in programming can be transferred to more general situations and can help the child to a more mature style of thinking and working.

### **ABOUT THE BOOK**

The book is arranged in 33 lessons, each with notes to the instructor and each containing assignments and review questions.

For instructors who feel themselves weak in BASIC or are beginners, the student's lessons form a good introduction to BASIC. The lessons and notes differ in style. The lessons are pragmatic and holistic, the notes and GLOSSARY are detailed and explanatory.

The book starts with a bare bones introduction to programming, leading quickly to the point where interesting programs can be written. See the notes for lesson 5, THE INPUT COMMAND, for an explanation. The central part of the book emphasizes more advanced and powerful commands. The final part of the book continues this, but also deals with broader aspects of the art of programming such as editing and debugging, and user friendly programming.

The assignments involve writing programs, usually short ones. Of course, many different programs are satisfactory "solutions" to these assignments. In the back of the book I have included a solution for each assignment program, some of them written by children who have used the book.

Lessons 14 (SAVE TO THE DISK) and 18 (EDIT AND RUN MODES) can be studied anytime after the first lesson.





# INTRODUCTION

## INSTRUCTOR NOTES 1 HOME, PRINT, NEW, and RUN

This lesson is an introduction to the computer. Directions for turning on the machine are given on the next page and may be supplemented by your own instructions.

Appendix A concerns care and usage of disks. You should initialize a disk at this point for the exclusive use of the student. Instructions and a useful HELLO program for this are given in the appendix.

Commands covered in this lesson are:

HOME, NEW, PRINT and RUN

The contents of the lesson:

1. Turning on the computer.
2. Typing versus entering commands or lines. RETURN key.
3. The computer only understands a limited number of commands.
4. In this lesson, HOME, NEW, PRINT, RUN.
5. What is a program. Numbered lines.
6. The screen can be cleared using HOME.
7. Memory can be cleared with NEW.
8. What is seen on the screen and what is in memory are different. This may be a hard concept for the student to understand at first.
9. RUN makes the computer go to memory, look at the commands in the lines (in order) and perform the commands.
10. One can skip numbers in choosing line numbers, and why one may want to do so.

### QUESTIONS:

1. Write a program that will print your name.
2. Make the program disappear from the TV screen but stay in memory.
3. Run it.
4. Erase the program from memory.
5. Write a program that will clear the screen, then print "HELLO"
6. Make it run.
7. Erase it from memory but leave it on the screen.

## LESSON 1 HOME, PRINT, NEW, and RUN

### HOW TO TURN ON THE COMPUTER

If you have the Apple II+ with Autostart ROM, then:

1. Find your special disk or the one named "DOS 3.3 SYSTEM MASTER".
2. Hold the disk with the label up and your thumb on the label.
3. Put the disk carefully in the drive 1 and close the door.
4. Reach behind the computer at the left and switch it on.
5. Switch on the TV or Monitor. If you have another kind of Apple, follow these instructions:

---

---

---

---

---

### TYPING

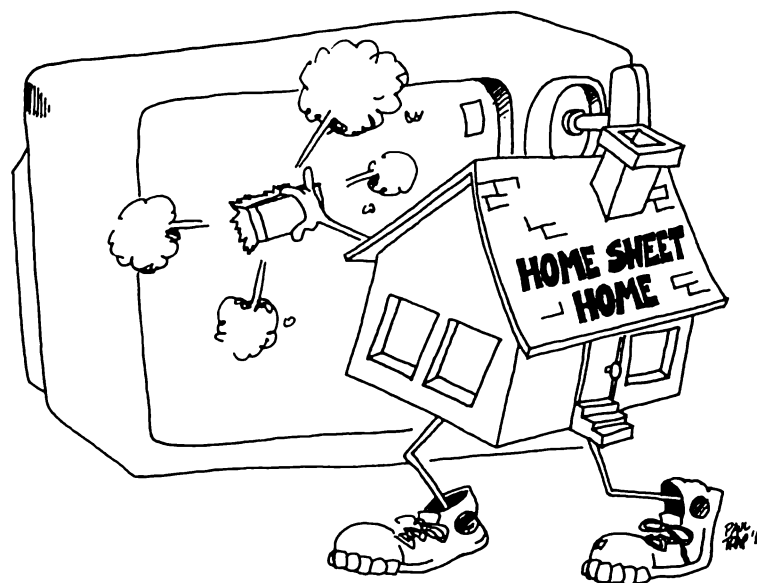
Type some things. What you type shows on the TV screen.

### COMMAND THE COMPUTER

Try this. Type: HOME

and press the RETURN key.

The word "HOME" tells the computer to erase the TV screen and move the flashing box to the upper left corner of the screen.



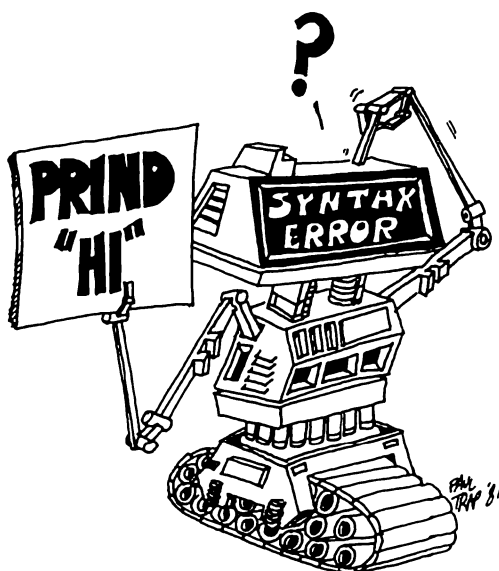
Did the computer “peep” at you and print SYNTAX ERROR? If so, type HOME and press the RETURN key again.

When the computer “peeps” and prints SYNTAX ERROR, it means the computer did not understand you.

The computer only understands about 100 words. You need to learn which words the computer understands.

Here are the first 4 words to learn:

HOME, NEW, PRINT, and RUN.

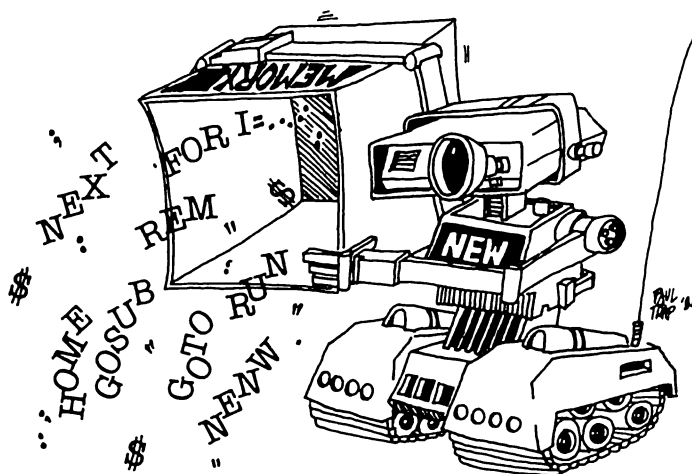


### THE NEW COMMAND

Type: NEW

and press RETURN.

NEW empties the computer’s memory so you can put your program in it.



## HOW TO ENTER A LINE

When we say, "enter" we will always mean to do these 2 things.

1. type a line
2. then press the RETURN key.

Enter this line:           10 PRINT "HI "

(The " marks are quotation marks. To make " marks, hold down the SHIFT key and press the key that has the 2 and the " on it.)

(Did you remember to press the RETURN key at the end of the line?)

Now the line number 10 is in the computer's memory. It will stay in memory until you enter the NEW command, or until you turn off the computer. Line 10 is a very short program.

## THE NUMBER ZERO AND THE LETTER "O"

The computer always writes the zero like this:

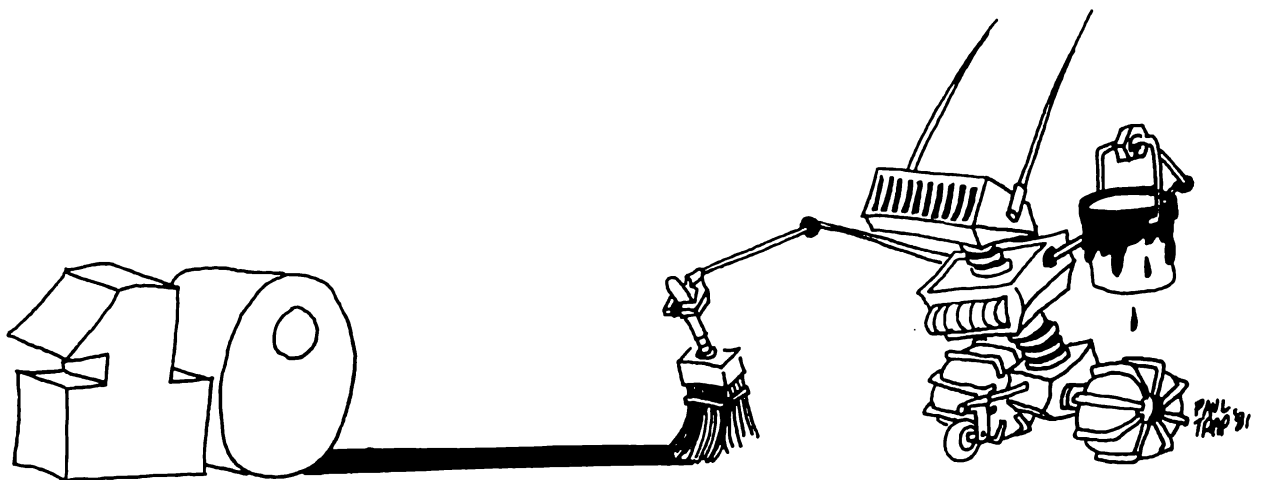
zero	0
------	---

and the letter O like this:

letter O	O
----------	---

You have to be careful to do the same.

right	10 HOME
wrong	1O HOME





## WHAT IS A PROGRAM?

A program is a list of commands you wish the computer to do. The commands are written in lines. Each line starts with a number. The program you entered above has only one line.



## HOW TO RUN A PROGRAM

A moment ago you put this program in memory:

```
10 PRINT "HI"
```

Now enter:                RUN

(Did you remember to press the RETURN key?)

The RUN command tells the computer to look in its memory for a program and then to obey the commands it reads in the lines.

Did the computer obey the PRINT command? The PRINT command tells the computer to print whatever is between the quotation marks. The computer printed:

HI



## HOW TO NUMBER THE LINES IN A PROGRAM

Enter this program:

```
1 REM PROGRAM 1
2 HOME
3 PRINT "HI "
```

This program has 3 lines. Each line starts with a command. You have already learned the HOME and PRINT commands. You will learn about the REM command later.

Usually you will skip numbers when writing the program.

Like this:

```
10 REM PROGRAM 1
15 HOME
20 PRINT "HI "
```

It is the same program but has different numbers. The numbers are in order, but some numbers are skipped. You skip numbers so that you can put new lines in between the old lines later if the program needs fixing.

Run the program you have entered. The computer does the commands in the lines. It starts with the lowest line number and goes down the list in order.

### Assignment 1:

1. Use the command HOME. Explain what it does.
2. Use the command NEW. Explain what it does.
3. Write a program that uses HOME once and PRINT twice. Then use the command RUN to make the program obey the commands.

## INSTRUCTOR NOTES 2 PEEPING, FLASH, INVERSE, AND NORMAL

This lesson opens with the `PRINT CHR$(7)` statement which makes the Apple “peep.” We wish to make plenty of “bells and whistles” available to the student to increase program richness.

The idea of a “string constant,” used in Lesson 1, is explained. The numbers appearing in a string, for example the “19,” cannot be used directly in arithmetic.

The `FLASH` and `INVERSE` commands put a little pizzazz on the screen. `NORMAL` puts the printing back to the familiar case. These commands can be used from the edit mode, or as statements in programs.

### QUESTIONS:

1. How do you do each of these things:  
Make the Apple “peep”?  
Erase the screen?  
Empty the memory?  
Print your name?
2. What is a “string”?
3. What special key do you press to “enter” a line?
4. What is a command? Give some examples.
5. What does the computer mean when it prints “SYNTAX ERROR”?
6. How could you print “FIRE” in flashing letters and make the computer peep?



## LESSON 2 PEEPING, FLASH, INVERSE, AND NORMAL

Enter: NEW  
Enter: HOME

NEW empties the memory and HOME erases the screen. You are ready to start this lesson.

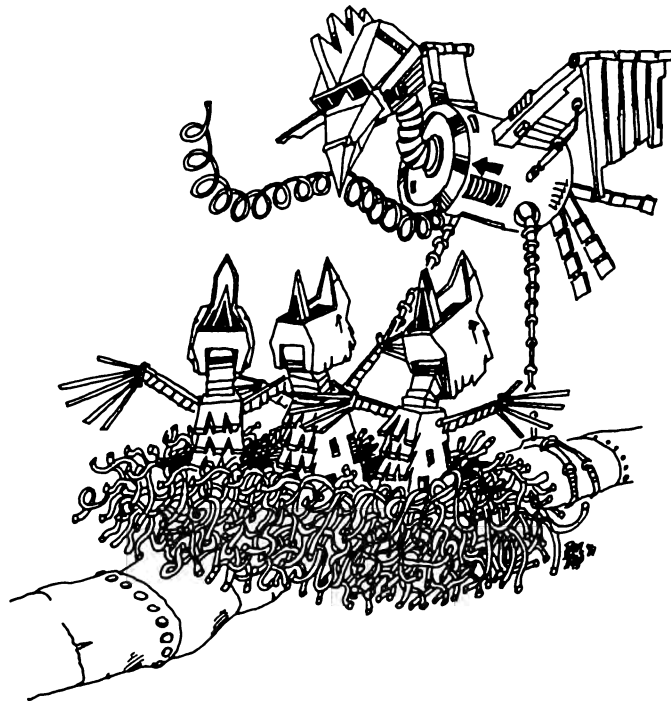
### THE APPLE PEEPS LIKE A BIRD.

Enter this program:

```
10 HOME  
20 PRINT CHR$(7)  
RUN
```

Did the Apple “peep”? Line 20 makes the apple peep. The number in the ( ) has to be 7.

You may want to have the Apple peep in some of your programs.



### PRINTING AN EMPTY LINE

Run this:

```
10 HOME  
20 PRINT "HERE IS A LINE"  
30 PRINT  
40 PRINT "ONE LINE WAS SKIPPED"
```

Line 30 just prints a blank line.

## STRING CONSTANTS

Look at these PRINT statements:

```
10 PRINT "JOE"  
10 PRINT "#D47[[*[["  
10 PRINT "19"  
10 PRINT "3.14159265"  
10 PRINT "I 'M 14"  
10 PRINT " "
```

Letters, numbers and punctuation marks are called “characters.” Even a blank space is a character. Look at this:

```
10 PRINT " "
```

Characters in a row make a “string.”

The letters are stretched out like beads on a string.

A string between quotation marks is called a “string constant.”

It is a string because it is made of letters, numbers and punctuation marks in a row.

It is a constant because it stays the same. It doesn't change as the program runs.



## FLASHY PRINTING

Use these commands to make the stuff printed on the screen look more interesting.

NORMAL is the usual kind of printing on the screen.

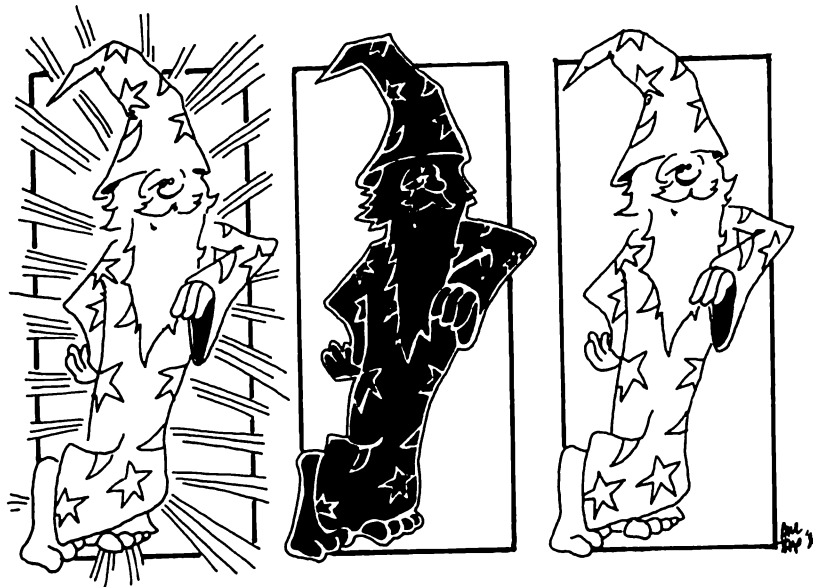
INVERSE has dark letters on a bright background.

FLASH has letters that blink.

**Important!** After using FLASH or INVERSE in a program, put NORMAL at the end so that printing is ok when the program is over.

Enter and run:

```
10 REM INVERSE , FLASH AND NORMAL
20 HOME
30 PRINT "HI THERE"
40 INVERSE
50 PRINT "INVERSE NOW"
60 NORMAL
70 PRINT "NORMAL AGAIN"
80 FLASH
85 PRINT "FLASHING NOW"
90 NORMAL
```



### Assignment 2:

1. Write a program that prints your first, middle and last names, with the first name flashing, the middle name inversed and the last name in the normal way.
2. Now add a "peep" before it prints each name.

## INSTRUCTOR NOTES 3 LIST AND REM

In this lesson:

- LIST, LIST 30
- REM for titles, remarks
- memory boxes holding lines
- erase one line from memory
- add a line between old lines
- replace a line
- drawings using PRINT commands

Actually, the difference between “command” and “statement” is artificial. The BASIC interpreter does not distinguish between them. Our wishes are called “commands” when used in the edit mode and “statements” when used in a program line. HOME is a good example of a command used both ways. In the first part of this book I will call all these things “commands” and later on explain what is meant by a statement (when talking about colons and having several statements on one line.)

For now your student needs to understand that the program is stored in memory even when it is not visible on the screen, and that LIST just lists the program to the screen. The special uses like LIST 100-300 and LIST -300 will be taken up later.

The memory as a shelf of boxes is a key model of the computer that we will develop in this book. It is an important tool in helping the student understand variables and the detailed workings of complicated expressions in a statement.

REM as a remark command can be a little confusing to new students. It needs to be distinguished both from PRINT and from just typing to the screen. Using print to draw pictures is demonstrated. It is better to draw some at the end of each lesson than to do a lot now. Drawing after lesson 4 helps develop line editing skills.

### QUESTIONS:

1. How do you erase a line you no longer want?
2. Type HOME. Now how do you show all of the program in memory on the screen?
3. How can you replace a wrong line with a corrected one?
4. Suppose you want to put a line in between two lines you already have in memory. How do you do this?
5. Explain how the computer puts program lines in “boxes” in memory. What does it write on the front of the box?

## LESSON 3 LIST AND REM

Enter:                   HOME  
                          NEW

Start each lesson with NEW and HOME to erase the screen and the memory.

Now enter:           10 HOME  
                         20 PRINT"LISTEN"  
                         30 PRINT CHR\$(7)  
                         40 PRINT"DID YOU HEAR IT?"

Run this 4 line program. Then enter HOME to erase the screen. The program is no longer visible on the screen.

But the program is not lost. The computer has stored the program in its memory. We can ask the computer to show us the program again.

### LISTING THE PROGRAM

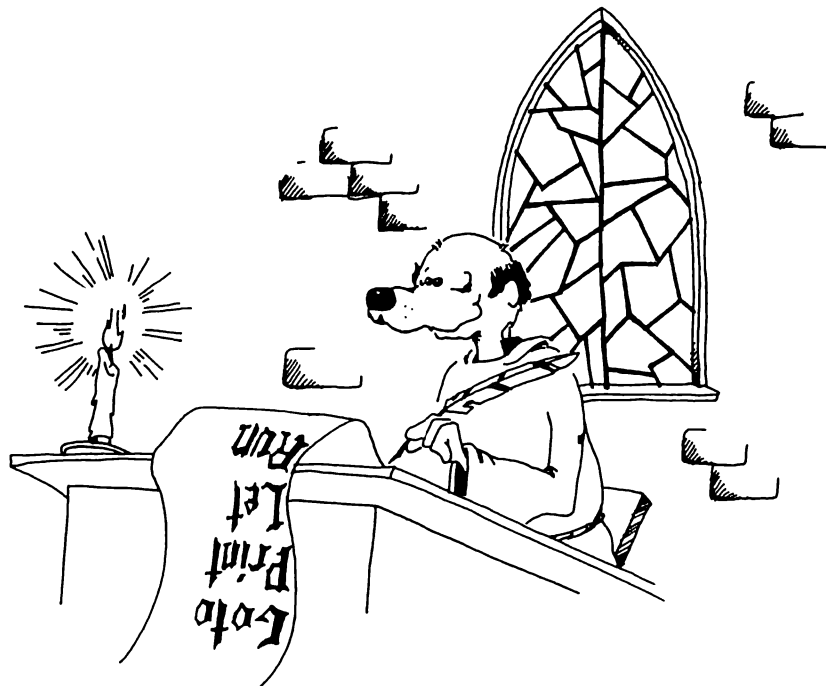
To ask the computer to show line 30 of the program, enter:

LIST30

The computer will list whatever line you ask for by number. If you want to list the whole stored program just enter

LIST

with no number after it. Try it.





## THE MEMORY

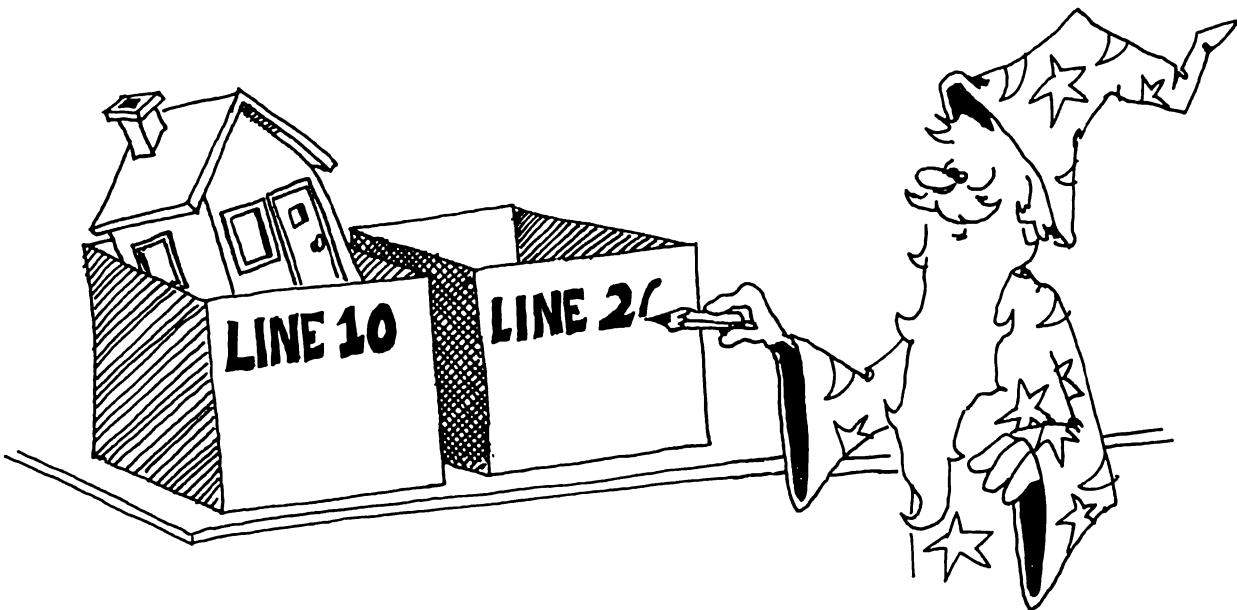
The computer's memory is like a shelf of boxes. The name of the box goes on the front of each box. At the start, all the boxes are empty and no box has a name.

When you entered: `10 HOME`

the computer took the first empty box and wrote the name "Line 10" on the label. Then it put the command HOME in the box and put the box back on the shelf.

When you entered: `20 PRINT "LISTEN"`

the computer took the second box and wrote "Line 20" on its label. Then it put the statement PRINT "LISTEN" in the box and put that box in its place on the shelf.



## ERASING A LINE FROM MEMORY

To erase one line of the program, enter the line number with nothing after it. For example, to erase line 20, enter:

`20`

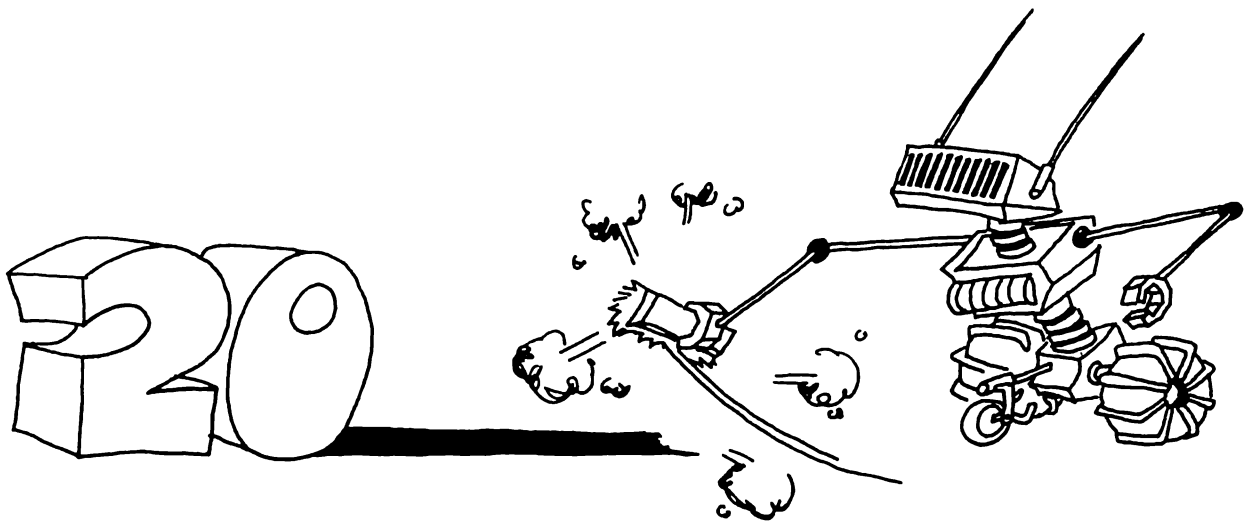
You still see the line on the screen, but do a LIST and you see that line 20 is gone from memory.

When you enter just a line number with nothing after it, the computer finds the box with that line number on it, empties the box and erases the name off the front of the box.

How do you erase the whole program? (Look at the beginning of this lesson to see the answer.)

---

What does the computer do to the boxes when you give it the command NEW?



### ADDING A LINE

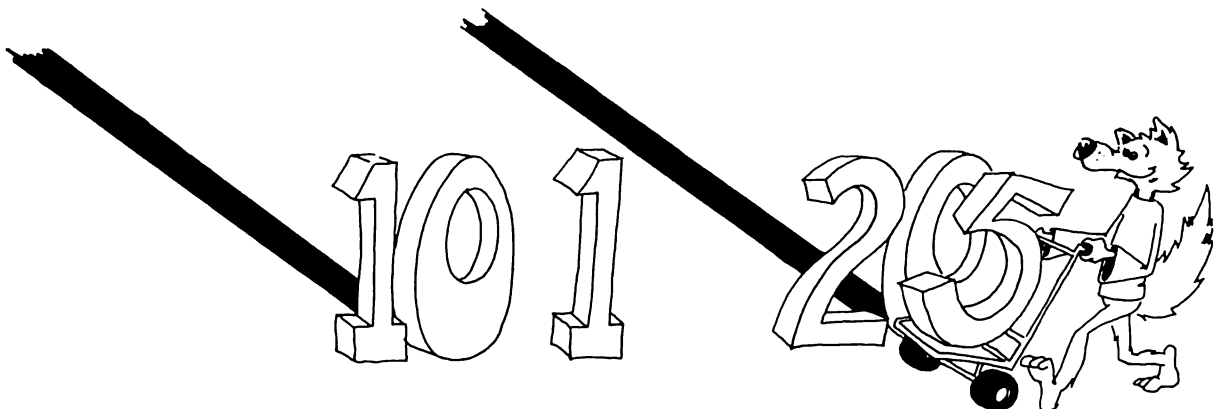
You can add a new line anywhere in the program, even between two old lines. Just pick a line number between those of the old lines, and type your line in. The computer puts it in the correct place.

Enter NEW and this:   10 REM MORE AND MORE  
                               20 PRINT "MORE LINES WANTED"  
                               40 PRINT "HERE THEY ARE"

List it and run it. Now add this line:

15 PRINT "STILL"

List and run it again.

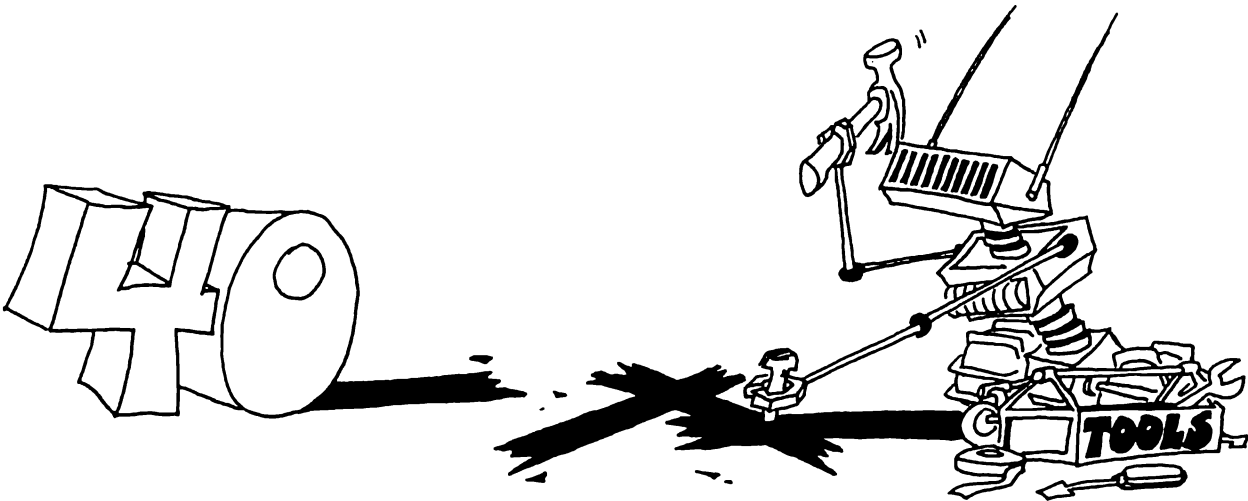


## FIXING A LINE

If a line is wrong, just type it over again. For example, in the above program line number 40 can be changed by entering:

```
40 PRINT "NEEDS FIXING"
```

What did the computer do to the box named "Line 40" when you entered the line?



## THE REM COMMAND

Use a REM command to put a title on your program.

Enter NEW and this:

```
10 REM PROGRAM 2
20 HOME
30 PRINT "LINE 10 DOES NOTHING"
35 REM THIS LINE DOES NOTHING
40 LIST
RUN
```

REM means "remark." Use REM to write any little note in the program that can help the reader understand the program.



### PICTURE DRAWING

You can use the PRINT command to draw pictures. Here is a picture of a car. Enter NEW before drawing the car.

```
10 HOME
20 PRINT
30 PRINT " XXXXXX"
40 PRINT "XXXXXXXXXXXXX"
50 PRINT " 0          0"
```

Don't forget to put the spaces in the PRINT lines! They are part of the drawing.

### Assignment 3:

1. What command will list line 10 of the program?
2. How do you tell the computer to list the whole program on the screen?
3. What does the computer do (if anything) when it sees the REM command?
4. What is the REM command used for?
5. Use HOME, CHR\$(7), REM, and PRINT to draw 3 flying birds on the screen. Make each bird peep after it is drawn.

## INSTRUCTOR NOTES 4 SPECIAL KEYS

This lesson concerns the arrow keys and the REPT key.

The arrow keys are used in moving the cursor around in the line currently being worked on. Characters in the line are not affected by the cursor moving over them. Wherever the cursor stops, you can type in new characters. Characters cannot be inserted or deleted, only replaced by others or by spaces. When all is satisfactory, the line can be entered in the computer by pressing RETURN, as usual. But . . .

**Warning:** Only the part of the line between start (at the left) and the current location of the cursor will be entered. The part under and to the right of the cursor will be discarded. If you want to save all the line, move the cursor to the end.

The arrow keys cannot be used alone to fix a line that had been entered earlier. A procedure for editing “old” lines will be given later.

The REPT key (repeat) is useful for moving the cursor fast (with the arrow keys), or making any repeated character, such as a line of dots (periods) in graphics.

### QUESTIONS:

1. What is a “cursor”? What is it good for?
2. What part of the line gets cut off if you press RETURN while the cursor is in the middle of a line?

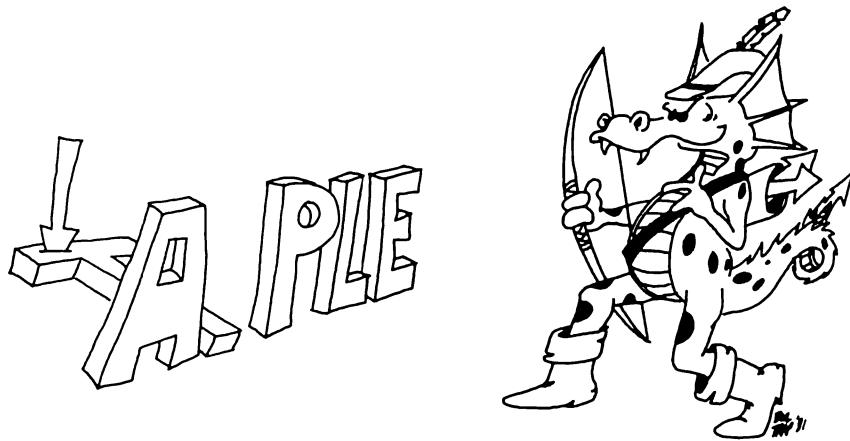
Have your student demonstrate:

3. How to edit a line. This includes using the arrow keys to move the cursor to the interior of the line, modifying characters there, and returning to the end before pressing RETURN.
4. Using the REPT key.

## LESSON 4 SPECIAL KEYS

### THE CURSOR IS A FLASHING SQUARE

The little flashing square is called the “input cursor.” It shows you where the next letter you type will appear on the screen. (Cursor means “runner.” The little square runs along the screen showing where the next letter will appear.)



### THE ARROW KEYS

The “arrow keys” help you fix errors in your typing.

Type: `10 REM ATPLE.` (do not press RETURN)

Press the left arrow key several times to move the cursor (the flashing square) over the “T.”

Type a “P” instead.

Use the right arrow to move the cursor to the end of the line (after the period).

Now the line is correct, reading:

`10 REM APPLE.`

Press RETURN to store the correct line in the memory.

## CAREFUL

Move the cursor back to the end of the line before pressing the RETURN key. If you forget, the end of the line will be chopped off! Try it!

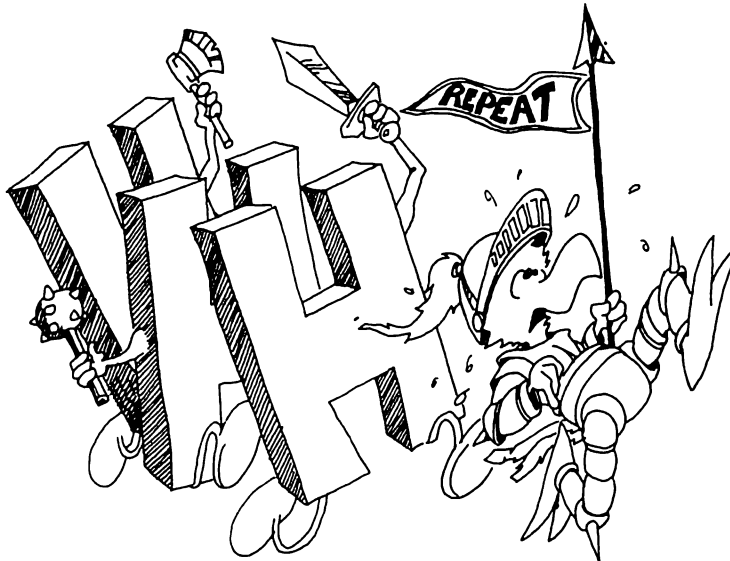


## REPEAT KEY

Press the REPT key and any letter key. What happens?

Press the REPT key and the left arrow key. What happens?

**Rule:** The REPT key can be used with any other key on the keyboard. Try it with the space bar and with the RETURN key.

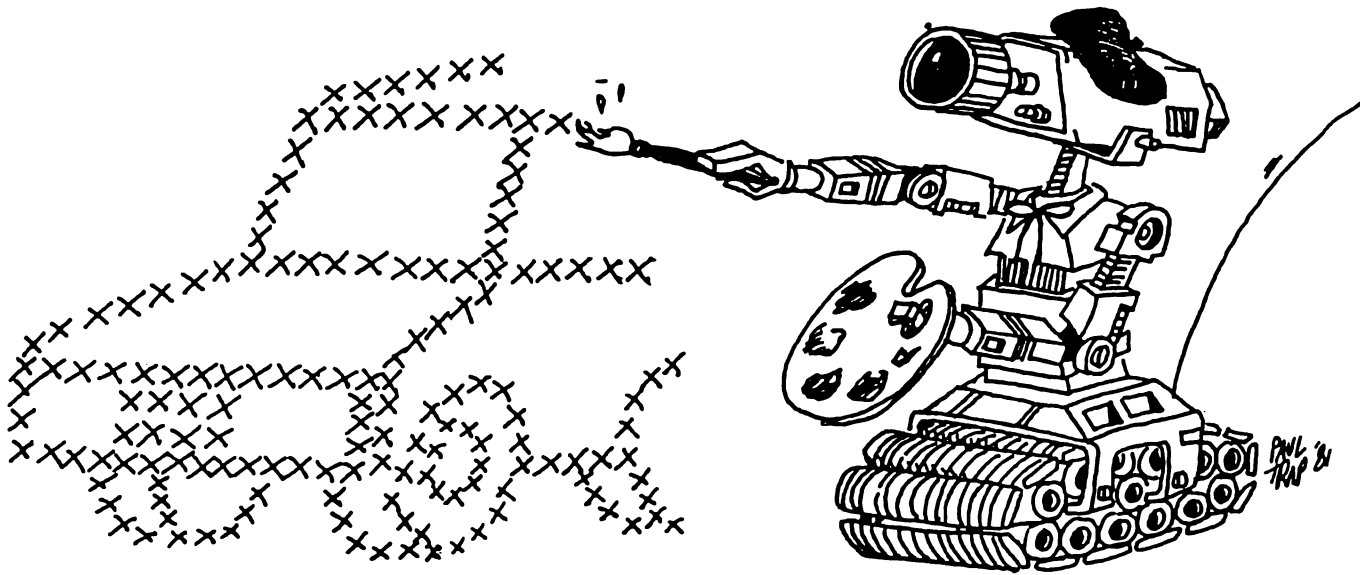


## DRAWING PICTURES

The REPT key is handy in drawing pictures. Use it to repeat letters or punctuation marks in the drawing.

### Assignment 4:

1. Type a line and use the arrow keys to move around in the line. Change letters in the line. When you are done, move the cursor to the end of the line and press RETURN to enter the line.
2. Draw a "smiley face."





## INSTRUCTOR NOTES 5 THE INPUT COMMAND

This lesson concerns the INPUT command and the idea of a string variable.

We introduce the input command in its simplest form:

INPUT A\$

that is, without a message in quotes in front. This allows the student to concentrate on the central feature of an INPUT.

Similarly, we will give only the essential feature of each command for the whole of the introduction of the book (through lesson 14). This allows us to quickly outline the essentials of programming so that the student “sees the forest” and is able to write meaningful programs. The commands required for interesting programs are:

PRINT	allows output
INPUT	input
GOTO	infinite looping
IF	branching and decisions
RND	random numbers for games

Back to this lesson. String variables are introduced using the “box” concept again. Variable names are restricted to one letter for the time being. This does not lead to confusion in short programs and allows faster typing. It also puts off the discussion of the complicated naming rules until after our sprint to the RND command.

We will work with strings and ignore numbers for as long as possible because strings make for more interesting programs and offer a less confusing entry into the logical concepts of programming.

### QUESTIONS:

1. What two different things does the computer put in boxes?
2. How does the program ask a person to type in something?
3. How do you know the computer is waiting for an answer?
4. A letter with a dollar sign after it is called what?
5. Write a short program that uses HOME, PRINT and INPUT.
6. Are you in trouble if the computer answers “EXTRA IGNORED” after an input? What made it do that?

## LESSON 5 THE INPUT COMMAND

Use INPUT to make the computer ask for something.

```
Enter:      10 HOME
            15 PRINT "SAY SOMETHING"
            20 INPUT A$
            25 PRINT
            30 PRINT "DID YOU SAY "
            40 PRINT A$
```

Run it. When you see a question mark, type "HI" and press the RETURN key.

The question mark was written by INPUT in line 20. The flashing cursor means the computer expects you to type something in.

When you type "HI," the computer stores this word in a box named A\$.

Later, in line 40, the program asks the computer to print whatever is in the box named A\$.

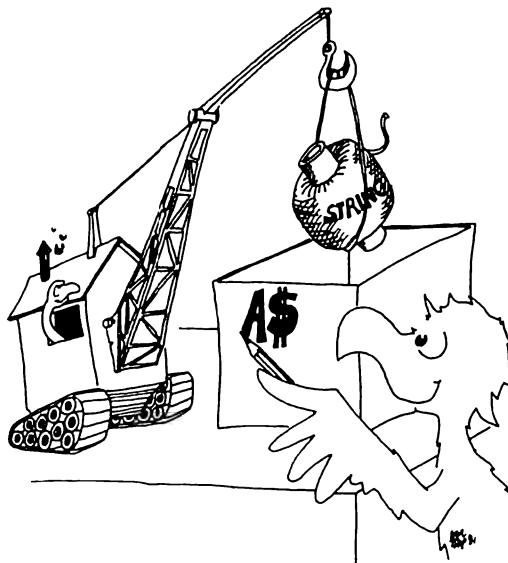
Run the program again and this time say something funny.

### STRING VARIABLES

A\$ is the name of a "string variable." The name is written on the front of a box and the string is put inside the box.

**Rule:** A string variable name always ends in a dollar sign, "\$." You can use any letter you like for the name and then put a dollar sign after it.

A\$ is called a variable because you can put different strings in the box at different times in the program. The box can hold only one string at a time.



## ERROR MESSAGES FROM INPUT

**Rule:** Do not put any commas, quotation marks, semicolons, or colons in the string you type in answer to the computer.

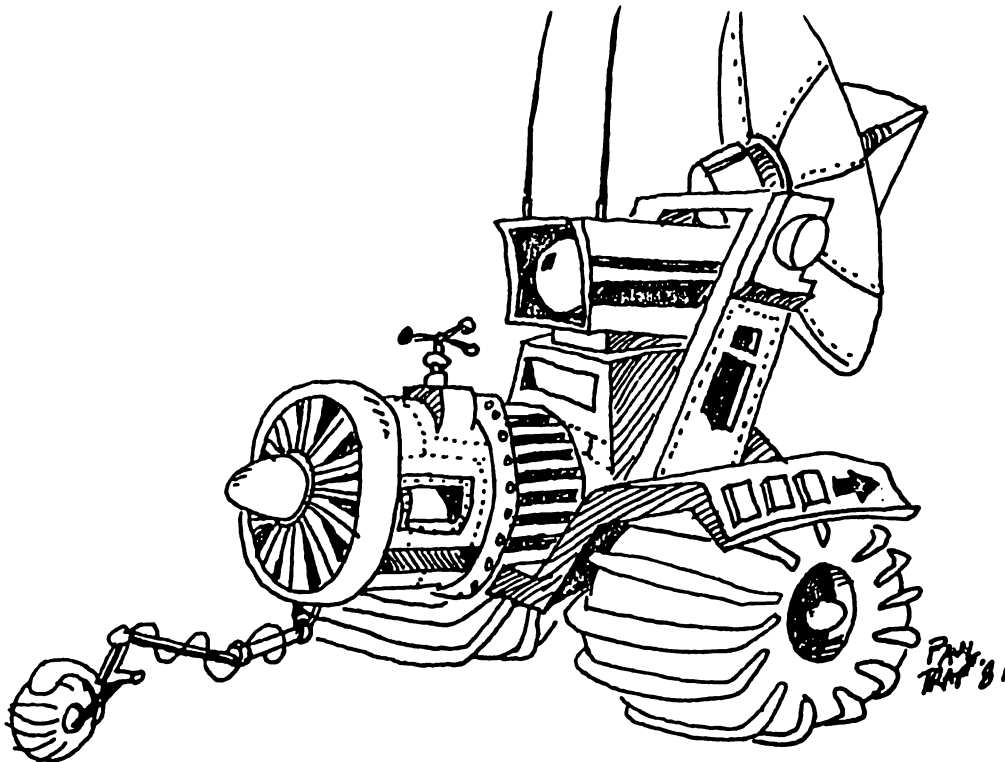
If you accidentally do put one in, the computer may answer:

?EXTRA IGNORED

and continue. This means that the computer chopped off everything after the comma (or " or ; or : mark), and then continued running the program.

### Assignment 5:

1. Write a program that asks for a person's name and then says something silly to the person, by name.
2. Write a program that asks you to INPUT your favorite color and put it in a box called C\$. Now the program asks you your favorite animal and puts this in box C\$ too. Have the program print C\$. What will be printed? Run the program and see if you are right.



## INSTRUCTOR NOTES 6 TRICKS WITH PRINT, SPEED

In this lesson:

PRINT with a semicolon at the end  
PRINT with semicolons between items  
the "invisible" PRINT cursor  
the SPEED command

The use of commas in PRINT is ignored as it is of little use on a 40-column screen.

The lesson introduces the output cursor which is invisible on the screen. It marks the place where the next character will be placed on the screen by a PRINT command. (The input cursor is the flashing square. It is familiar from the edit mode and the INPUT command.)

When a PRINT statement ends with a semicolon, the output cursor remains in place and the next PRINT will put its first character exactly in the spot following the last character printed by the current PRINT command.

Without a semicolon at the end, the PRINT command will advance the output cursor to the beginning of the next line as its last official act.

A PRINT command can print several items, a mixture of string and numerical constants, variables, and expressions. Numerical constants and variables have not yet been introduced. The items are separated by semicolons.

The series of printed items will have their characters in contact. If spaces are desired, as in the "HAM AND EGGS" example, the spaces have to be put in the strings explicitly.

### QUESTIONS:

1. Which cursor is a little flashing square? What command puts it on the screen?
2. Which cursor is invisible? What command uses it?
3. How do you make two PRINT statements print on the same line?
4. Will these two words have a space between them when run?

```
10 PRINT "HI" ; "THERE !"
```

If not, how do you put a space between them?

5. What command causes the printing to be very slow?
6. How do you get the printing speed back to normal?

## LESSON 6 TRICKS WITH PRINT, SPEED

### ONE LINE OR MANY?

Enter this program:

```
10 REM FOOD
20 PRINT
30 PRINT "HAM"
40 PRINT "AND"
50 PRINT "EGGS"
```

and run it. Each PRINT command prints a separate line.

Now enter:

```
30 PRINT "HAM ";
40 PRINT "AND ";
```

(Don't change or erase the other lines.) Be careful to put the space at the end of "HAM " and at the end of " AND " and the semicolon at the end of each line.

Run it.

What was different from the first time?

---

---

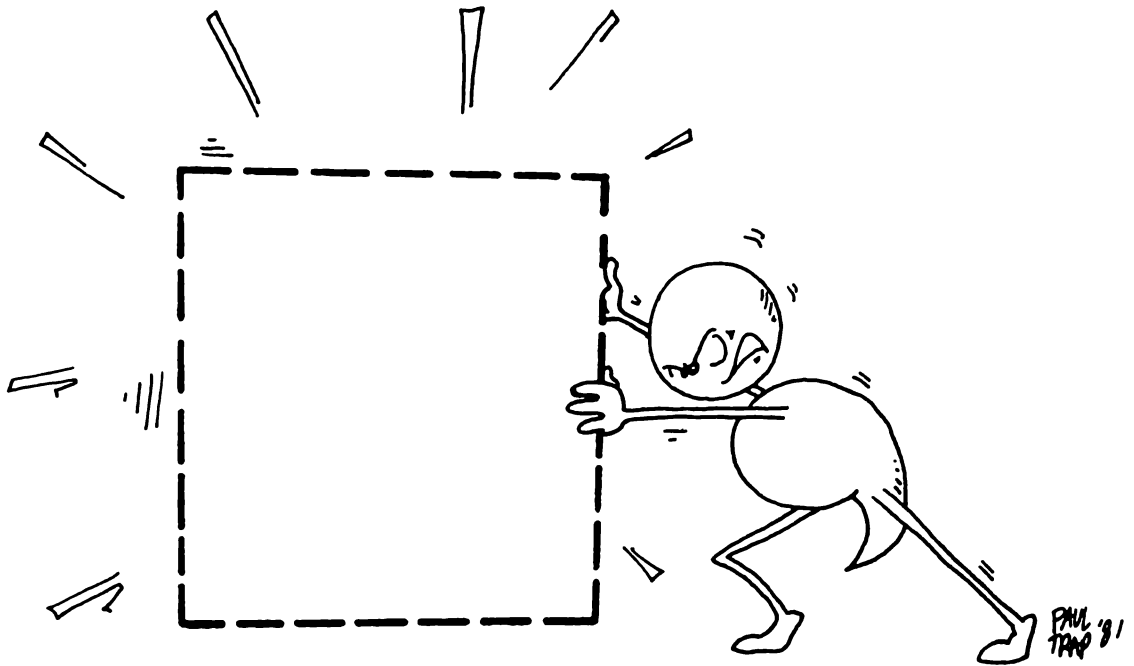
### THE HIDDEN CURSOR

Remember the flashing square? It is the INPUT cursor and shows where the next letter will appear on the screen when you type.

The PRINT command also has a cursor, but it is invisible. It marks where the next letter will appear when the computer is PRINTing.



**Rule:** The semicolon makes the invisible PRINT cursor wait in place on the screen. The next PRINT command adds on to what has already been written on the same line.



### FAMOUS PAIRS

Enter:

```
10 HOME
20 PRINT "ENTER A NAME"
30 INPUT A$
35 HOME
40 PRINT "ENTER ANOTHER"
50 INPUT B$
60 HOME
70 PRINT "PRESENTING THAT FAMOUS TWOSOME"
75 PRINT
80 PRINT A$;" AND ";"B$
```

Be sure to put a space before and after the "AND."

### SQUASHED TOGETHER OR SPREAD OUT?

Enter NEW then try this:

```
10 PRINT "ROCK";"AND";"ROLL"
```

after you have run it, try also:

```
10 PRINT "ROCK "; "AND "; "ROLL"
```

don't forget the spaces after ROCK and AND.

## THE SPEED COMMAND

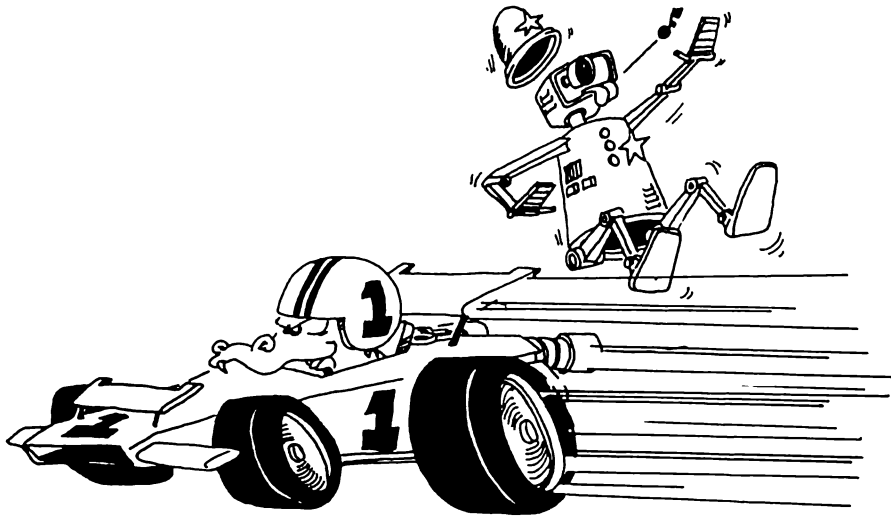
Sometimes the usual printing speed of the computer is too fast. The **SPEED** command slows the printing.

Enter and run:

```
10 REM :::::SPEED:::::  
20 HOME  
30 PRINT"MISSISSIPPI"  
40 SPEED=5  
50 PRINT"MISSISSIPPI"  
60 SPEED=255
```

The speed can be set to any number from 1 to 255. **SPEED = 1** is very slow. **SPEED = 255** is the normal speed.

Look at line 60. It is very important. It puts the computer back to normal speed when the program is done.



## Assignment 6

1. Write a program that asks for the name of a musical group and one of their tunes. Then using just one **PRINT** command, print the group name and the tune name, with the word "plays" in between.
2. Do the same, but use 3 print commands to print on one line.
3. Have the computer print your name slowly in the inverse mode.

## INSTRUCTOR NOTES 7 THE LET COMMAND

The LET command is introduced using the concept of memory boxes.

The box model is used to emphasize that LET is a replacement command, not an “equal” relationship in the sense used in arithmetic.

The box idea nicely separates the concepts “name of the variable” and “value of the variable.” The name is on the label of the box, the value is inside. The contents of the box may be removed for use, and new contents inserted.

More exactly, a copy of the contents is made and used when a variable is used, the original contents remain intact. This point is explained.

Concatenation is covered under the heading “gluing the strings.”

Used so far:

HOME, NEW, PRINT, RUN, PRINT CHR\$(7), FLASH, INVERSE, NORMAL, LIST, REM, INPUT, SPEED, LET

Special keys discussed so far:

RETURN, REPT, two arrow keys and the shift key.

### QUESTIONS:

1. LET puts things in boxes. So does INPUT. How are they different?

2. If you run this little program:

```
10 LET A$ = "HI "  
20 LET B$ = A$
```

what will be in box A\$ at the end? What will be in box B\$?

3. In this program:

```
10 Q$ = "MOM"
```

what is “MOM” called? What is the name of the string variable in this program? What is the value of the string variable after the program runs?

4. What is in each box after this program runs?

```
10 LET H$ = "FAT"  
20 LET K$ = " SAUSAGE "  
30 LET P$ = A$ + K$
```



## LESSON 7 THE LET COMMAND

The LET command puts things in boxes. Enter and run:

```
10 HOME
20 LET W$="MOPSEY"
40 PRINT W$
```

Here is what the computer does:

Line 10 The computer clears the screen.

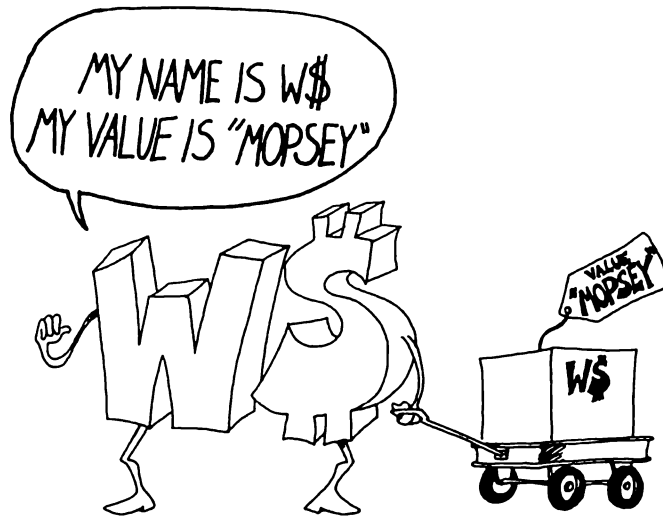
Line 20 It sees that a box named "W\$" is needed. It looks in its memory for it. It doesn't find one because "W\$" has not been used in this program before. So it takes an empty box and writes "W\$" on the front, and then puts the string "MOPSEY" in it.

Line 40 The computer sees that it must print whatever is in box "W\$." It goes to the box and makes a copy of the string "MOPSEY" that it finds there. It puts the copy on the TV screen. The string "MOPSEY" is still in box "W\$."



### NAMES AND VALUES

The name of the variable was W\$. The value of the variable is put in the box. In the program above, the value of W\$ is "MOPSEY."



### ANOTHER EXAMPLE:

Enter and run:

```
10 LET D$="PICKLES"
20 LET A$=" AND "
30 PRINT "WHAT GOES WITH PICKLES?"
35 INPUT Z$
40 HOME
50 PRINT D$;A$;Z$
```

Explain what the computer does in each line.

- 10 \_\_\_\_\_
- 20 \_\_\_\_\_
- 30 \_\_\_\_\_
- 35 \_\_\_\_\_
- 40 \_\_\_\_\_
- 50 \_\_\_\_\_



## GLUING THE STRINGS

Here is how to stick two strings together to make a longer string. Enter:

```
10 HOME
20 LET W$="HAR DE "
25 LET X$="HAR "
30 L$=W$+X$
40 PRINT L$
50 PRINT
60 LET L$=L$+X$
70 PRINT L$
```

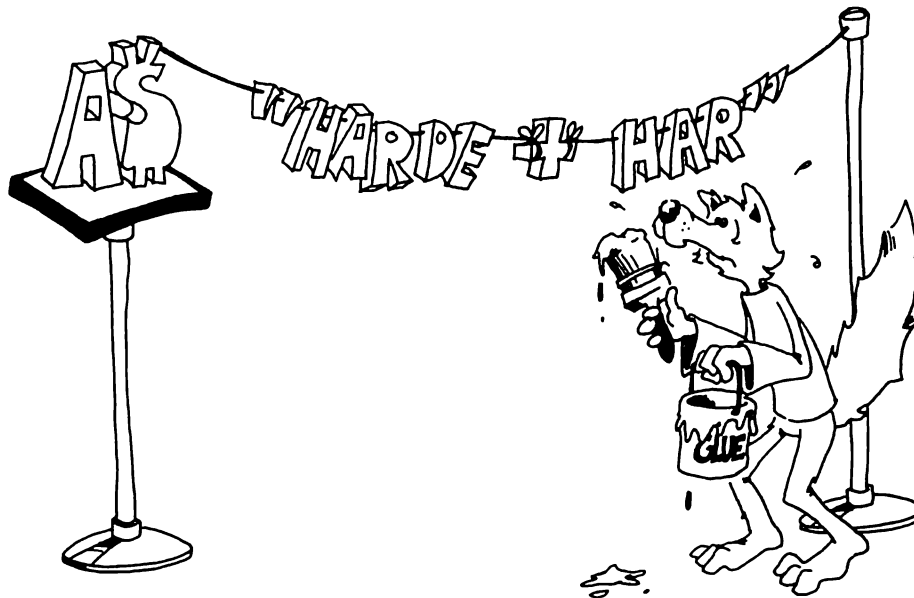
Before you RUN this program, try to guess what will be printed at line 40 and at line 70:

40 \_\_\_\_\_

70 \_\_\_\_\_

Now run the program to see if you were right.

**Rule:** The “+” sign sticks two strings together. Notice that line 50 in the program above just prints an empty line. This is often good for spacing apart the things your program must print.



### Assignment: 7

1. Write your own program that uses the LET command and explain how it stores things in “boxes.”
2. Write a program that inputs two strings, glues them together and then prints them.

## INSTRUCTOR NOTES 8 THE GOTO COMMAND AND RESET KEY

The GOTO command allows a “dumb” loop that goes on forever. It also helps in flow of command in later programs, after the IF is introduced. It provides a slow and easy entrance for the student into the idea that the flow of command need not just go down the list of numbered lines.

For now its main use is to let programs run on for a reasonable length of time. In each loop through, something can be modified.

The problem is how to stop it. The RESET key does this nicely, if you have an Apple II with autostart ROM. If you have an Apple II Plus, get your pen out and modify the lesson to read “CTRL-C” everywhere that RESET is written. On some Apple II Plus, the CTRL key has to be pressed together with RESET to do a reset.

We now have three of the four major elements that lead to “real” programming. They are PRINT, INPUT, and GOTO. Lacking is the IF, which will change the computer from some sort of a record player into a machine that can evaluate situations and make decisions accordingly.

### QUESTIONS:

1. In this little program:

```
10 PRINT "HI"  
20 GOTO 40  
30 PRINT "BIG"  
40 PRINT "DADDY"
```

what will appear in the screen when it is run?

2. And this one:

```
10 PRINT "APPLE"  
20 PRINT "PIE ";  
30 GOTO 20
```

3. How do you stop the program in question 2?
4. Write a short program that “peeps,” asks you your favorite movie star’s name, and then does it over and over again.

## LESSON 8 THE GOTO COMMAND AND RESET KEY

### JUMPING AROUND IN YOUR PROGRAM

Try this program:

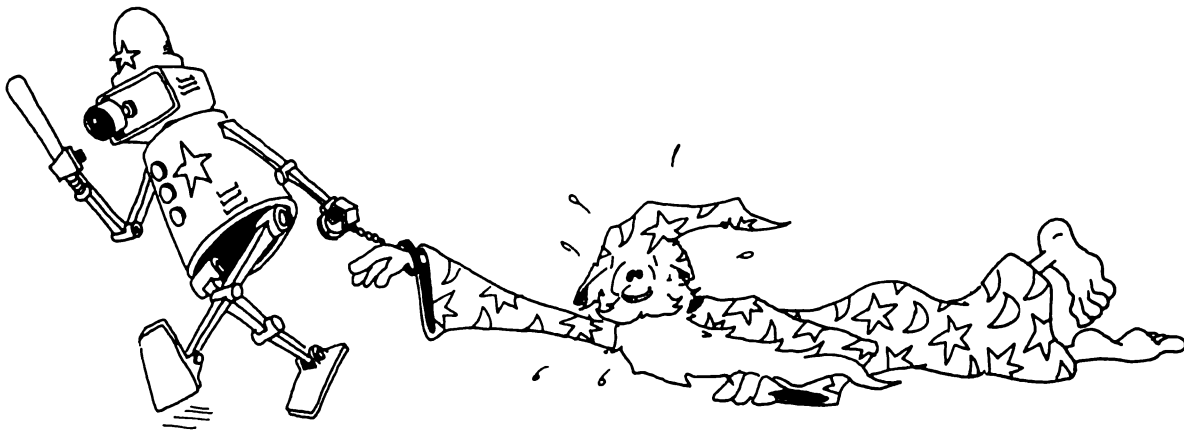
```
10 HOME
20 PRINT "YOUR NAME?"
25 INPUT N$
30 PRINT N$
35 PRINT
40 GOTO 30
```

RUN this program. It never stops by itself! To stop your name from whizzing past your eyes, press the

CONTROL and RESET key at the same time. (or CONTROL C)

Line 40 uses the GOTO command. It is like "GO TO JAIL" in a game of Monopoly. Every time the computer reaches line 40, it has to go back to line 30 and print your name again.

We will use GOTO in a lot of programs.



### MORE JUMPING

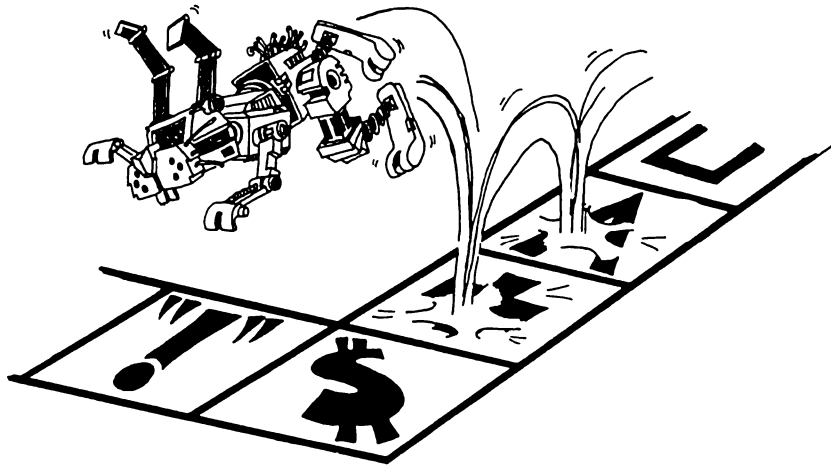
Enter:

```
20 PRINT "SAY SOMETHING"
30 INPUT S$
40 PRINT "DID YOU SAY '" ; S$ ; "'?"
45 PRINT
50 GOTO 30
```

An arrow points from line 50 to line 30, indicating the jump.

Run the program. Type an answer every time you see the "?" and the flashing cursor. Press the RESET key to end the program.

Notice the arrow from line 50 to line 30. It shows what the GOTO does. You may want to draw such arrows in your program listings.



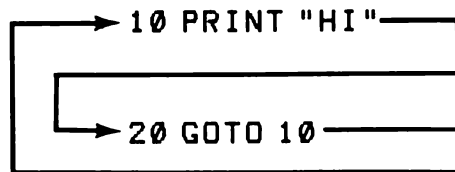
## KINDS OF JUMPS

There are only two ways to jump: ahead or back.

Jumping back gives a LOOP.

```
10 PRINT "HI"  
20 GOTO 10
```

The path through the program is like this:

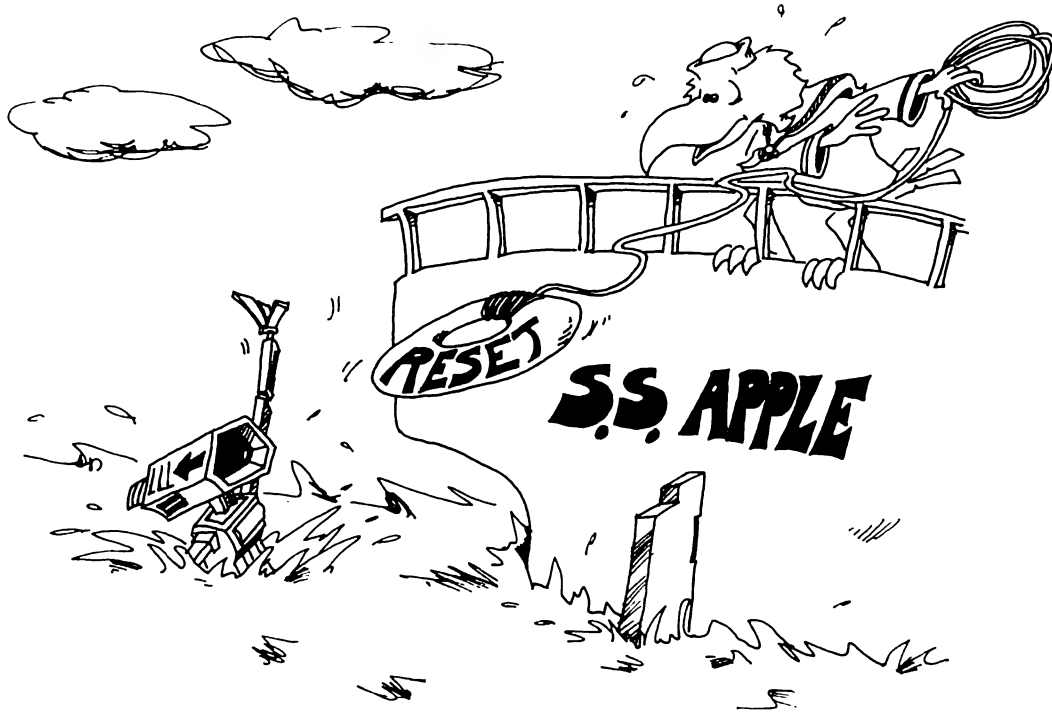


The computer goes around and around in this loop. Press the RESET key to stop.

Jumping ahead lets you skip part of the program. It is not useful yet, but we will use it later in the IF command.

## THE RESET

The CONTROL C RESET key is a “life saver.” When you are in trouble, press CONTROL C or RESET and the computer will “peep” and start over, waiting for your next command. Your program is still safe in memory.



### Assignment 8:

1. Write a program that prints your first name over and over.
2. How do you stop your program?
3. Write another that prints your name on one line, then a friend's on the next, over and over. Use `SPEED` to slow down the printing. Stop the program with the `RESET` key, then enter `SPEED = 255` to get the printing speed back to normal.
4. Write a program that uses each of these commands: `HOME`, `PRINT`, `CHR$(7)`, `PRINT`, `INPUT`, `LET`, and `GOTO`. It also should glue two strings together.

## INSTRUCTOR NOTES 9 THE IF COMMAND

The IF command is introduced in this lesson. The case where two strings are the same or not the same is treated.

IF is a powerful command that is at the very heart of the computer as a logic machine. It is an intricate command and the student may require extra help at this point.

The IF command appeals both to our verbal and our visual imagination. The “cake” cartoon and the “fork in the road” cartoon illustrate these ideas. That the flow of commands may be altered has already been introduced with the GOTO command. To that idea is now added the conditional test: if an expression is true, one thing happens, if it is false, another.

The phrase “something A” is used for the expression being tested for truth. The Apple manuals call “something A” an “assertion.” The phrase “command C” is used for the command to be done if the assertion is true.

Two ideas occurring in the “something A” may be confused by the student. They are the “=” relation and the truth of the overall assertion.

The case where the “< >” sign is used may help distinguish the idea of the “truth” of the assertion from the logic within the assertion.

On the other hand, the “< >” sign is probably unfamiliar to the student and it is possible that confusion is compounded at this point. In that case, just work with the overall IF idea and use the “=” case for examples. We will return to the IF at later points in the book and at that time familiarity with IF will work to the student’s advantage. The larger set of relations:

<,      >,      =,      =<,      =>,      <=>

will be treated then.

### QUESTIONS:

1. How do you make this program print “THAT’S FINE”?

```
15 PRINT "DOES YOUR TOE HURT?"
17 INPUT T$
20 IF T$="NAH" THEN GOTO 90
40 IF T$="SOME" THEN GOTO 15
90 PRINT "THAT 'S FINE"
```

2. Write a short program which asks if you like chocolate or vanilla ice cream. Answers to be “C” or “V.” For the “C” print “Yummy!”. For the “V” answer, print “Mmmmmm!”.



## LESSON 9 THE IF COMMAND

Clear the memory and enter:

```
10 HOME
20 PRINT "ARE YOU HAPPY? (YES OR NO)"
30 INPUT A$
40 IF A$="YES" THEN PRINT "I'M GLAD"
50 IF A$="NO" THEN PRINT "TOO BAD"
```

Run the program several times. Try answering "YES," "NO" or "MAYBE." What happens?

YES \_\_\_\_\_

NO \_\_\_\_\_

MAYBE \_\_\_\_\_

### THE IF STATEMENT

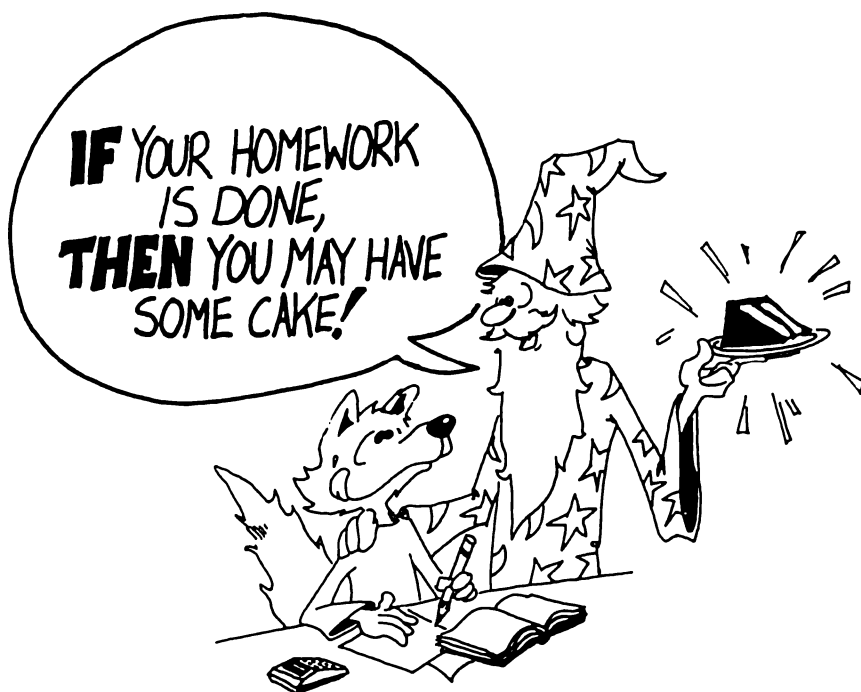
The IF statement has two parts:

10 IF something A THEN command C

First the computer looks at "something A."

If it is true, the computer does the command C.

If "something A" is not true, then the computer goes on to the next line without doing the command C.



It looks like this:

```
10 IF    something A is true    THEN    do command C
      and then go on to the next line
```

or

```
10 IF    something A is false    THEN
      go on to the next line
```

### SOME EXAMPLES OF IF

```
10 REM IF TEST PROGRAM
50 IF  A$="YES"           THEN  PRINT "GOOD"
55 IF  "YES"=A$           THEN  PRINT "GOOD"
60 IF  A$=B$              THEN  LET C$="NO WAY!"
70 IF  N$="BIRD"           THEN  PRINT CHR$(7)
75 IF  A$="READY"         THEN  INVERSE
```

Line 50 and line 55 do exactly the same thing.

### Assignment 9A:

1. Add these lines to the program:

```
15 INPUT A$
17 LET C$="WHAT?"
20 LET B$="PAY UP"
25 LET N$=A$
85 PRINT C$
87 NORMAL
99 GOTO 15
```

Run the program and enter these words:

YES , BIRD , READY , NO WAY

and some other words you choose yourself. Look at what the program prints to see that IF commands are working as you expect. (Remember, if "something A" is true, then the command after the THEN is executed.)

2. Clear memory and write a program that asks if you are a "BOY" or "GIRL." If the answer is "BOY," the program prints "SNIPS AND SNAILS." If the answer is "GIRL," print "SUGAR AND SPICE."

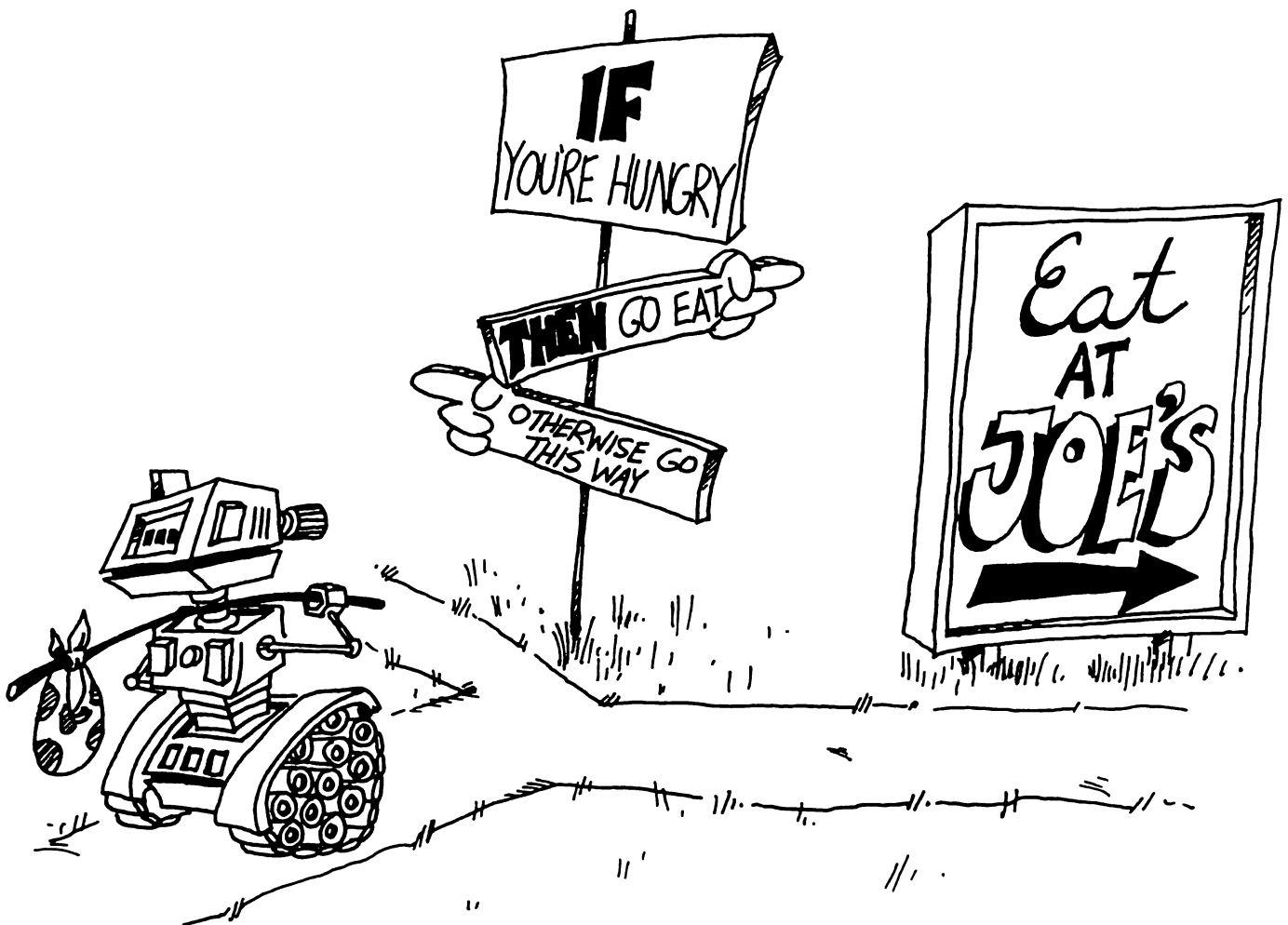
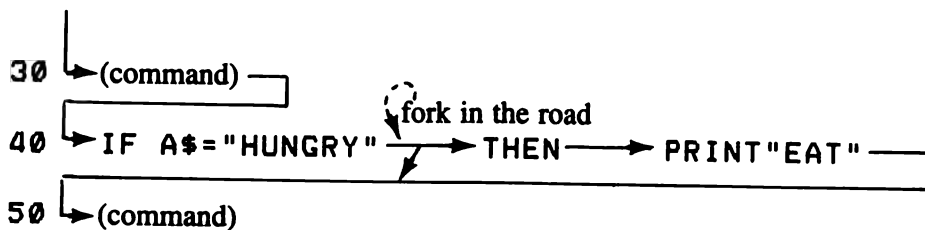
## A FORK IN THE ROAD

When it sees "IF," the computer must choose which road to take.

If "something A" is true, it must go past the "THEN" and obey the command it finds there. Then it goes down to the next line.

If "something A" is false, it goes down to the next line right away.

Here is the road map with the fork in the road marked:



## THE "NOT EQUAL" SIGN

The "< >" sign means "not equal." It is the opposite of the "=" sign when used in an English phrase.

To make the "< >" sign, press the "<" key, then the ">" key.

```
40 IF something A      THEN      PRINT "NO SMOKE"
```

"Something A" is a phrase that is TRUE or FALSE. If it is true, then the computer prints "NO SMOKE." Look at this "something A" phrase:

```
Q$<>"FIRE"
```

and put it in an IF command:

```
40 IF  Q$<>"FIRE"      THEN      PRINT "NO SMOKE"
```

If the Q\$ box contains "COLD" then Q\$ is not equal to "FIRE" and the expression Q\$<>"FIRE" is TRUE. The computer will print "NO SMOKE."

If the Q\$ box contains "FIRE" then the phrase:

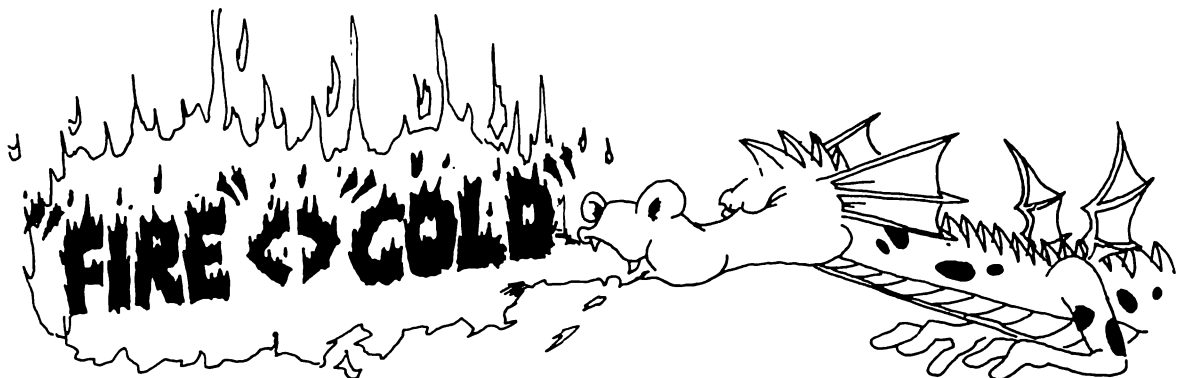
```
Q$ <> FIRE      is FALSE  and
```

computer will not print anything.

Here is how it looks in a program:

```
10 PRINT"IS YOUR HOUSE ON FIRE?"
20 PRINT"(ENTER 'FIRE' OR 'COLD')"
```

```
30 INPUT Q$
40 IF Q$<>"FIRE" THEN PRINT "NO SMOKE"
50 IF Q$= "FIRE" THEN PRINT "HELP"
```

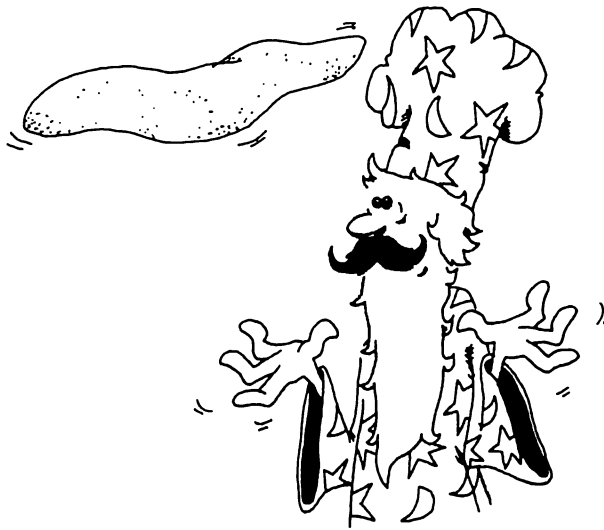


### Assignment 9B:

1. Write a "pizza" program. Ask what topping is wanted. Make the computer answer something silly for each different choice. You can choose mushrooms, pepperoni, anchovies, green peppers, etc. You can also ask what size.
2. Write a color guessing game. One player INPUTs a color in string C\$ and the other keeps INPUTing guesses in string G\$. Use two IF lines, one with a "something A"

G\$< >C\$

for when the guess is wrong, and the other with an "=" sign for when the guess is right. The "command C" prints "wrong" or "right."



## INSTRUCTOR NOTES 10 INTRODUCING NUMBERS

Numerical variables and operations are introduced. The LET, INPUT, and PRINT commands are revisited.

The idea of memory as a shelf of “boxes” is extended to numbers. Again, variable names are limited to one letter for the time being.

The arithmetic operations are illustrated. The “\*” symbol for multiplication will probably be unfamiliar to the student. Division will give decimal numbers, so it is nice if your student is familiar with them. But most arithmetic will be addition and subtraction, with a little multiplication, and a student unfamiliar with decimal numbers will not experience any disadvantage.

It may seem strange to the student that the numbers in string constants are not “numbers” that can be used directly in arithmetic. The VAL and STR\$ functions will be introduced later in the book and allow interconversion of numbers and strings.

A mixture of string and numerical values can be printed by PRINT. The non-standard use of “=” in BASIC, that it means “replace” and not “equal”, shows up strongly in the statement:

LET N = N + 1

The cartoon uses the box idea to illustrate this meaning of “=”.

### QUESTIONS:

1. Name the three kinds of “boxes” in memory. (That is, named by the kinds of things stored in the boxes.)
2. Explain why “N = N + 1” for a computer is not like “7 = 7 + 1” in arithmetic.
3. Give another example of “bad arithmetic” in a LET command. Use the \*, or / symbols.
4. What does the computer mean by “TYPE MISMATCH ERROR”?
5. Give an example of a program line that would have a TYPE MISMATCH ERROR.
6. Explain what is meant by the “name of a variable” and the “value of a variable” for numerical variables. For string variables.
7. If the boxes A and B have the numbers 5 and 8 in them, then PRINT A;B will print 58. How do you make the computer print 5 8 instead of 58?

## LESSON 10 INTRODUCING NUMBERS

### INPUT, LET, AND PRINT

So far we have only used strings. Numbers can be used too. Enter and run this program:

```
10 HOME
20 PRINT "GIVE ME A NUMBER"
30 INPUT N
40 LET A=N+1
45 PRINT
50 PRINT "HERE IS A BIGGER ONE"
60 PRINT A
```

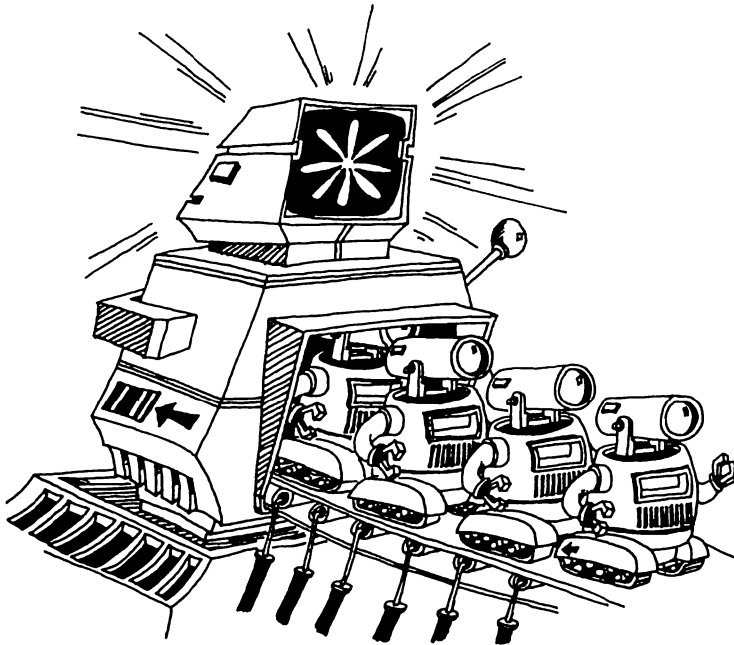
### ARITHMETIC

The minus sign is on the same key as the "=" sign.

Computers use "\*" instead of "×" for a multiplication sign.

Try this. Change line 40 so that N is multiplied by 5.

Computers use "/" for a division sign. Answers are given as decimals.

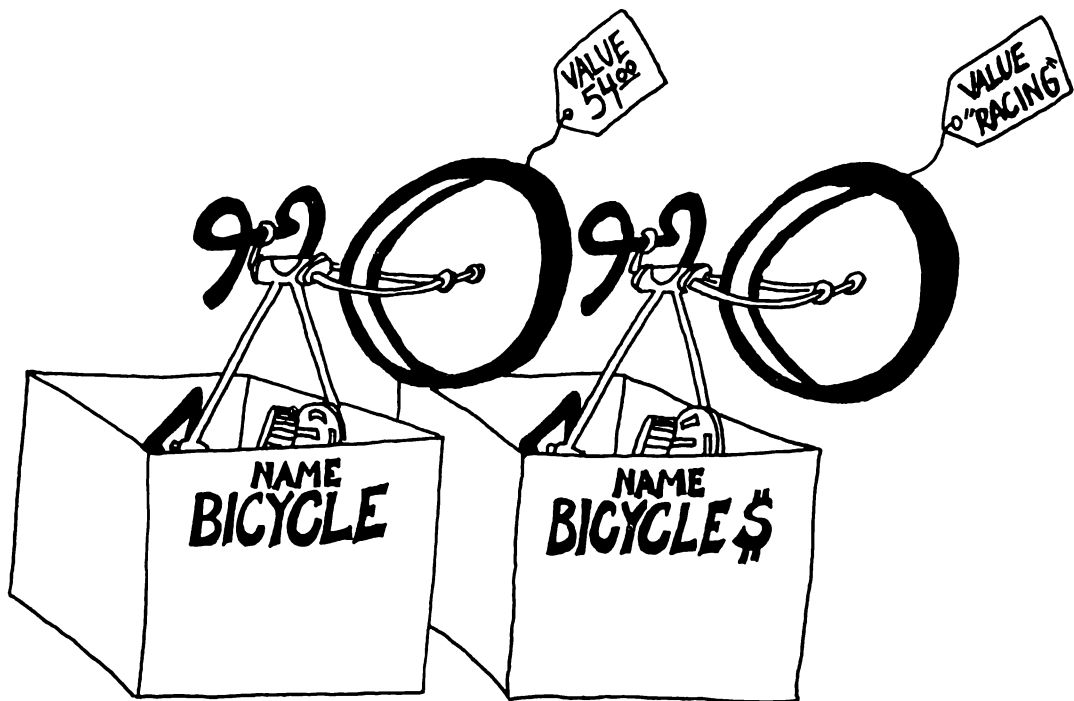


## VARIABLES

The name of a box that contains a string must end with a dollar sign. Examples: N\$, A\$, Z\$.

The name of a box that contains a number doesn't have a dollar sign. Examples: N, A, Z.

The thing that is put in the box is called the "value" of the variable.



## ARITHMETIC IN THE LET COMMAND

```
10 LET A=2
20 LET B=3
30 LET C=B-A
40 PRINT A;" ";B;" ";C
```

Some more examples:

```
10 LET B=15
20 LET A=B/5
30 LET X=A*4+2
40 PRINT X;" ";A
```



## CAREFUL!

Numbers and strings are different. Example: "1984" is not a number. It is a string constant because it is in quotes.

**Rule:** Even if a string is made up of number characters it is still not a number.

Some numerical constants: 5, 22, 3.14, -50

Some string constants: "HI", "7", "TWO", "3.14"

**Rule:** You cannot do arithmetic with the numbers in strings.

Correct:                   10 LET A = 3 + 7

Wrong:                   10 LET A\$ = 3 + 7

Wrong:                   10 LET A = "3" + "7"



If you run either of these wrong lines, the computer will print:

TYPE MISMATCH ERROR IN LINE 10

The two types of variables are "string" and "numerical." You cannot mix them.

Enter:                   10 LET A=5  
                         20 LET B\$="10"  
                         30 LET C=A+B\$

Lines 10 and 20 are ok, line 30 is wrong. What will the computer do when you run this little program? Try it.

Try to guess what each of these statements will print, then enter the line to see what happens:

PRINT 5	_____
PRINT "5"	_____
PRINT "5+3"	_____
PRINT "5"+"3"	_____
PRINT 5 + 3	_____

### MIXTURES IN PRINT

You can print numbers and strings in the same PRINT command. (Just remember that you cannot do arithmetic with the mixture.)

Correct:                   PRINT A;"SEVEN";"7"  
  
                          PRINT A;B\$

Run this line.           10 PRINT 5/2;" IS EQUAL TO 5/2"

### A FUNNY THING ABOUT THE EQUAL SIGN

The "=" sign in computing does not mean "equals" exactly. Look at this program:

10 LET N=N+1

This does not make sense in arithmetic. Suppose N is 7. This would say that:

7 = 7 + 1

which is not correct.

But it is ok in computing to say  $N = N + 1$  because the “=” sign really means “replace.” Here is what happens:

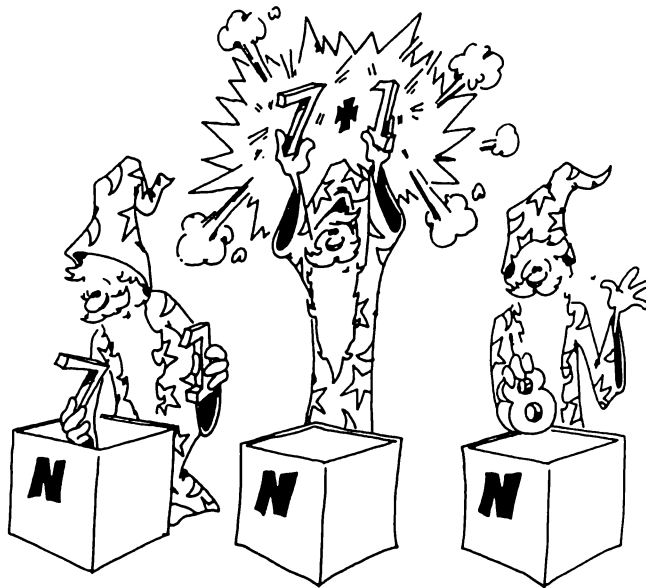
Look at this:           10 LET  $N = N + 1$

The computer goes to the box with N written on the front.

It takes the number 7 from the box.

It adds 1 to the 7 to get 8.

Then it puts the 8 in the box.



Another way to say the same thing is:

10 LET  $N = N + 1$  means

LET (new N) equal (old N) plus one

### Assignment 10:

1. Write a program that asks for your age and the current year. Then subtract and print out the year of your birth. Be sure to use PRINT statements to tell what is wanted and what the final number means.
2. Write a program that asks for two numbers and then prints out their product. (Multiplies them.)

## INSTRUCTOR NOTES 11 TAB AND DELAY LOOPS

The TAB command adds flexibility to the screen display. Delay loops slow the program down so that its operation can be more easily observed. They also are used for portions of the program that must run at certain speeds, and should then be called "timing loops."

TAB is used in a PRINT command and is like the tab on a typewriter. It is much more powerful than you might realize at first. It allows moving graphics (using strings) and interesting displays of verbal information.

Students who are not rather familiar with a typewriter may need extra help in seeing what a TAB is good for.

Several TAB commands can be used in one PRINT statement, but the arguments in the ( ) must increase each time. That is, TAB cannot be used to move the cursor back to the left. Later we treat the HTAB and VTAB commands which allow placement of the cursor anywhere on the screen.

Use of a semicolon between TAB and the thing to be printed is not always necessary, but is recommended.

This lesson introduces loops in a painless way.

The delay loop is all on one line, with a colon to separate off the NEXT command. The amount of delay is determined by the size of the loop variable. A value of 1000 gives about a one-second delay.

After seeing that the primary work of the loop is simply to count until a particular value is reached before going on to the next instruction, it will be easier for the student to handle loops in which things are going on inside.

### QUESTIONS:

1. Show how to write a delay loop that lasts for about 2 seconds.
2. Will this work for a delay loop?

```
120 FOR Q=1000 TO 5000
122 NEXT Q
```

3. Tell what the computer will do in each case:

```
10 PRINT "HI";TAB(20);"GOOD LOOKING!"
10 TAB(5);PRINT "OH-OH!"
10 PRINT TAB(15);"NOP";TAB(1);"NOT HERE"
```

## LESSON 11 TAB AND DELAY LOOPS

### THE TAB COMMAND

TAB in a PRINT command is like the TAB on a typewriter. It moves the printing cursor a number of spaces to the right.

(The printing cursor is invisible.)

The next thing to be printed goes where the cursor is.

Try this:

```
10 PRINT "123456789ABCDEF"  
20 PRINT 1;TAB(5);2  
30 PRINT TAB(3);"Y";TAB(9);"Z"
```

**Rule:** After TAB(N), then the next character will be printed in column N.

### CAREFUL!

Run this:

```
10 TAB(5)
```

You see SYNTAX ERROR IN 10. TAB( ) has to be in a PRINT command. You cannot use TAB( ) by itself.

### YOU CANNOT TAB BACKWARDS

Try this:

```
10 PRINT "123456789ABCDEF"  
20 PRINT 1;TAB(9);9;TAB(3);3
```

The TAB( ) command can only move the printing to the right. You cannot move back to the left.

### YOUR NAME IS FALLING!

```
10 HOME  
15 LET N=1  
20 PRINT"YOUR FIRST NAME"  
30 INPUT W$  
40 PRINT TAB(N);W$  
50 LET N=N+1  
60 GO TO 40
```

Press CONTROL RESET to stop the run.

This program prints your name in a diagonal down the screen, top left to bottom right. Try other values of N. Try changing lines:

```
15 LET N=30  
50 LET N=N-1
```

## HOW BIG A SPACE CAN TAB( ) MAKE?

There are 40 spaces across the screen. You can use any number 1 through 40 inside the TAB( ) parentheses. Larger numbers make the computer skip lines. Numbers larger than 255 will give an error message when the program runs:

ILLEGAL QUANTITY ERROR IN XX.

where XX is the line number.

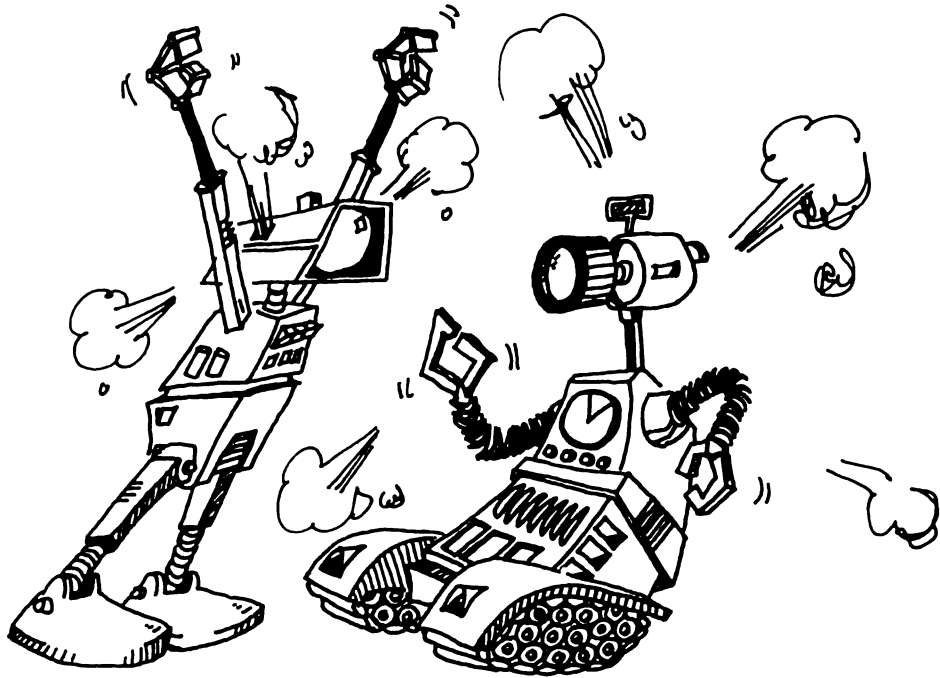
You can use TAB with strings too:

Example: `10 PRINT F$;TAB(15);M$;TAB(25);L$`

Here F\$, M\$, and L\$ are the strings for the first, middle, and last names.

## FUNCTIONS DON'T FIGHT BUT THEY HAVE ARGUMENTS

TAB( ) is a command that is like a "function." We will study other functions like RND( ), INT( ), LEFT\$( ), etc. The number inside the ( ) is called "the argument of the function." TAB( ) says "move the cursor over" and the argument tells "where to move it to."



### Assignment: 11A

1. Write a program that asks for last names and nicknames. Then print the last name starting at column 5 and the nickname at column 25. Use a GOTO so the program is ready for another name-age pair.
2. Write an "insult" program. It asks your name. Then it peeps, and writes your name. Then it TABS over in the line and prints an insult.

## DELAY LOOPS

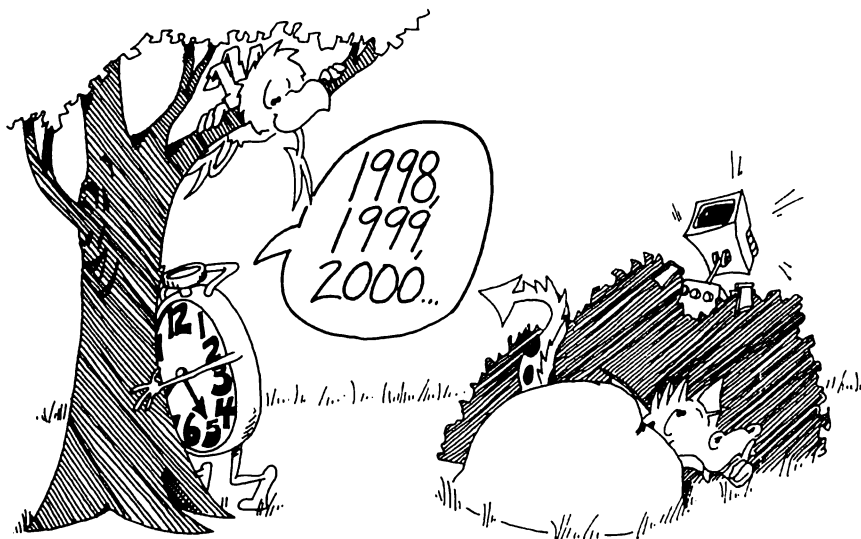
Remember the **SPEED** command? It lets you print much slower than normal.

Here is a way to slow down other parts of the program. It is a "delay loop."

Run this program:

```
10 REM DELAY LOOP
20 HOME
30 PRINT "WAIT"
40 FOR I=1 TO 2000:NEXT I
50 PRINT "DONE"
```

Line 40 is the delay loop. The computer counts from 1 to 2000 before going on to the next line. It is like counting when you are "it" in a game of hide and seek.



Try changing the number "2000" in line 40 to some other number.

Each 740 in the delay loop is worth about 1 second of time. Try this:

```
10 REM ---- TICK TOCK ----
20 HOME
30 INPUT "WAIT HOW LONG? "; S
36 T=S*740
40 FOR I=1 TO T:NEXT I
45 PRINT : PRINT CHR$(7)
50 PRINT S;" SECONDS ARE UP"
```

### Assignment 11B:

1. Write a "slow poke" program that prints out a three word message with several seconds between each word. Have the computer peep before each word.

## INSTRUCTOR NOTES 12 THE IF COMMAND WITH NUMBERS

The IF command is extended to numerical expressions. The logical relations used in this lesson are:

=,       >,       <,       < >

The use of nested IF's is demonstrated.

A "home made" loop is demonstrated in the GUESSING GAME, but not discussed. The loop starts in line 50 and goes to 80. The exit test is made in line 70. The logic of this loop is that of a DO UNTIL.

### QUESTIONS:

1. What part of the IF command can be TRUE or FALSE?
2. What follows the THEN in an IF command?
3. After this little program runs, what will be in box D?

```
10 LET D=4  
15 IF 3 < 7 THEN LET D=9
```

4. Same question, but for  $3 > 7$ .



## LESSON 12 THE IF COMMAND WITH NUMBERS

Try this:

```
10 REM *** TEENAGER ***
15 HOME
20 PRINT "YOUR AGE?"
30 INPUT A
40 IF A<13 THEN PRINT " NOT YET A TEENAGER! "
50 IF A>19 THEN PRINT " GROWN UP ALREADY! "
```

This IF command is like the one that you used before with strings. Again we have:

IF something A is true THEN do command C

"Something A" can have these arithmetic symbols:

=	equal to
>	greater than
<	less than
< >	not equal to

Each "something A" is a phrase. It is written in "math language" but you should say it out loud in English. For example:

$A < > B$  is pronounced "A is not equal to B"

$5 < 7$  is pronounced "five is less than seven"

### PRACTICE

For these examples, LET A = 7, LET B = 5, and LET C = 5.

Say each "something A" out loud and tell if it is true or false:

A = B	T	F
A > B	T	F
A < B	T	F
A = C	T	F
A < C	T	F
A > C	T	F
B = C	T	F
B > C	T	F
B < C	T	F
A < > B	T	F
B < > C	T	F

## AN IF INSIDE AN IF

The "teenager" program above is missing something. Add:

```
60 IF A>12 THEN IF A<20 THEN PRINT "TEENAGER!"
```

To understand this, break it into two parts:

```
60 IF A>12 THEN      (command C)      where
```

```
(command C)      is      ( IF A<20 THEN PRINT "TEENAGER!" )
```

This line first asks "is the age greater than 12?"

If the answer is "yes" the line gets to ask the second question: "Is the age less than 20?"

If the answer is again "yes" the line prints "TEENAGER!"


If the answer to either question is no, the PRINT command is not reached, so nothing is printed.

## Assignment 12A:

1. Draw the "fork in the road" diagram for line 60 above. There will be two forks on the diagram. (See page 1-45.)

## GUESSING GAME

```
10 REM --- GUESSING GAME ---
15 HOME
20 PRINT "TWO PLAYER GAME"
25 PRINT
30 PRINT "FIRST PLAYER ENTER A NUMBER FROM 1 TO 100"
35 PRINT "WHILE SECOND PLAYER ISN'T LOOKING"
37 PRINT
40 INPUT N
45 HOME
50 PRINT TAB(12); "MAKE A GUESS ";
55 INPUT G
60 IF G<N THEN PRINT "TOO SMALL"
65 IF G>N THEN PRINT "TOO BIG"
70 IF G=N THEN GOTO 90
80 GOTO 50
90 REM THE GAME IS OVER
92 PRINT
93 FLASH
95 PRINT "THAT'S IT!"
99 NORMAL
```



If you want to save this program on a disk, read Lesson 14.

Usually line 80 sends you to line 50 so you can make more guesses. But if  $G=N$  in line 70, then you skip to line 90 and print "THAT'S IT!"

## Assignment 12B:

1. Tell what happens in lines 50 through 80:

If G is 31 and N is 88:

50 \_\_\_\_\_

55 \_\_\_\_\_

60 \_\_\_\_\_

65 \_\_\_\_\_

70 \_\_\_\_\_

80 \_\_\_\_\_

If G is 88 and N is 88:

50 \_\_\_\_\_

55 \_\_\_\_\_

60 \_\_\_\_\_

65 \_\_\_\_\_

70 \_\_\_\_\_

80 \_\_\_\_\_



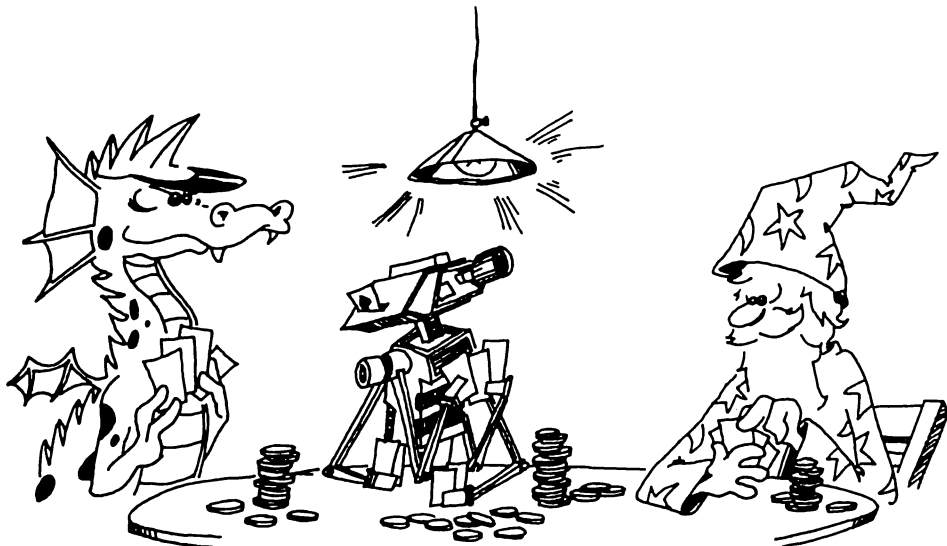
2. Here is another program. What will it print, and how many times?

```
10 LET N=1
20 IF N=13 THEN PRINT "UNLUCKY!"
30 LET N=N+2
40 IF N>30 THEN GOTO 99
50 GOTO 20
99 PRINT "DONE"
```

What will it print if line 10 is changed to:

```
10 LET N=2
```

3. Write a program that says something about each number from one to ten. The player enters a number and the computer prints something about each number: "three strikes, you're out" or "seven is lucky" etc.
4. Write a game for guessing a card that someone has entered. You must enter the suit (club, diamond, heart, or spade) and the value (1 through 13). First they guess the suit, then the program goes on to ask the value. Keep score.
5. Write a digital clock program. It uses a timing loop to count seconds. Input the present time in hours, minutes and seconds. The clock then counts seconds and prints them out. When 60 seconds have gone by, add one to the minutes and put seconds back to zero. Same with hours. Run the clock a long time and adjust the timing loop so the clock keeps good time.



## INSTRUCTOR NOTES 13   RANDOM NUMBERS AND THE INT FUNCTION

This lesson introduces two functions: RND and INT. These are very important in games and also handy in making interesting displays like kaleidoscopes.

The RND function produces psuedo-random decimal numbers between 0.0 and 1.0. Such numbers are directly usable as probabilities, but integers over some range such as 1 to 6 for a die, or 1 to 13 for a suit of cards are often more to the point.

Your student may be shaky in decimal arithmetic, but all that is required here is multiplication of the random number by an integer, and perhaps also addition to an integer. The computer does the multiplication, of course, so only a rough idea of the desired result is necessary.

After extending the random number to a larger range than 0 to 1, conversion to an integer is desired. The INT function does this by simply truncating the number, “throwing away the decimal part.” (For negative numbers the situation is a little more complicated, and that rare case is not treated here).

The concept of functions is again used in this lesson and is further clarified.

The nesting of one function in the parentheses of another is illustrated by using RND in the argument of an INT function.

### QUESTIONS:

1. Tell what the computer will print for each case:

`10 PRINT INT(G)`

and the “box G” contains: 2, 2.1, 2.95, 3.001, 67, 0, 0.2

2. Tell how the INT( ) function is different from “rounding off” numbers. Which is easier for you to do?
3. Tell how to change a number so that the INT( ) function will round it off.
4. What does the RND(8) function do?
5. How can you get random integers (whole numbers) from 0 through 10. (Hint: INT(RND(8)\*10) is not quite right.)
6. How can you get random integers from 5 through 8?

## LESSON 13 RANDOM NUMBERS AND THE INT FUNCTION

### THE RND FUNCTION

When you throw dice, you can't predict what numbers will come up.

When dealing cards, you can't predict what cards each person will get.

The computer needs some way to let you "roll dice" and "deal cards" and do many other unpredictable things.

Use the RND function to do this. RND stands for "random."

Run this program:

```
10 REM RANDOM NUMBERS
20 HOME
25 LET N=RND(8)
30 PRINT N
40 IF N<.95 THEN GOTO 25
```

You see a lot of decimal numbers on the screen. The RND function in line 25 made them.

It doesn't matter what number you put in the parentheses just so long as it is positive. I choose "8" because it is near the "( )" signs on the keyboard making it easy to type (8).



RND gives numbers that are decimals larger than 0 but smaller than 1. To make numbers larger than one, you just multiply.

Change the program above to:

```
25 LET N=RND(8)*52
40 IF N<45 THEN GOTO 25
```

and run it again.

Now the numbers are between 0 and 52 in size. They could be used for choosing the 52 cards in a deck.

But:

We usually want whole numbers like 7 and 8 rather than decimal numbers like 7.03454323 and 8.89746582. Do this by using the INT function.

### THE INT FUNCTION

The INT function takes the number in its parentheses and throws away the decimal part, leaving an integer.

Try the INT function in this little program:

```
10 LET I=INT (6.3)
20 PRINT I
```

And in this:

```
10 LET X=0.3
20 PRINT "X  =  ";X;TAB(10);"INT(X)= ";INT(X)
```

And this:

```
10 LET X=.3
20 LET Y=2.5
30 LET P=X+Y
40 LET Q=INT(X+Y)
50 PRINT P,Q
```

Look at the answers to see that the decimal part was thrown away.

Try this:

```
10 REM ----- INT -----
20 HOME
30 PRINT"GIVE ME A DECIMAL NUMBER "
32 INPUT D
35 LET I=INT(D)
40 PRINT "DECIMAL ";D;TAB(20);"INTEGER ";I
50 IF I<>0 THEN GOTO 30
```

Enter 0 to end the program.



### ROLLING THE BONES

Usually dice games use two dice. One of them is called a "die." Here is a program that acts like rolling a single die:

```
10 REM ////////// ONE DIE //////////
20 HOME
30 LET R=RND(8)
40 PRINT "RANDOM NUMBER";TAB(20);R
50 LET S=R*6
55 PRINT "TIMES 6";TAB(20);S
60 LET I=INT(S)
65 PRINT "INTEGER PART";TAB(20);I
70 LET D=I+1
75 PRINT "DIE SHOWS";TAB(20);D
77 PRINT
80 PRINT "ANOTHER? <Y/N> "
82 INPUT Y$
85 IF Y$="Y" THEN GOTO 20
```



### WHAT GOES INSIDE THE ( ) ?

Numbers:                10 LET X=INT(34,7)

Variables:             10 LET X=INT(J)

Expressions:          10 LET X=INT(3\*Y+2)

Functions:            10 LET X=INT(RND(8))

Here is how to save a lot of room.

Instead of:            30 LET R=RND(8)  
                          50 LET S=R\*6  
                          60 LET I=INT(S)  
                          70 LET D=1+I

Use just:              70 LET D=1+INT(RND(8)\*6)

### Assignment 13:

1. Write a program that “rolls” two dice, called D1 and D2. Show the number on D1 and on D2 and the sum of the dice. You do not need the variables R, S, and I in the program above. They were used to show how the final answer was found.
2. Write a “paper, scissors, and rock” game, you against the computer. (Paper wraps rock, rock breaks scissors, scissors cut paper). The computer chooses a number 1, 2 or 3 using the RND( ) function: 1 is paper, 2 is rock, 3 is scissors. You INPUT your choice as P, R, or S and the computer figures out who won and keeps score.

## INSTRUCTOR NOTES 14 SAVE TO THE DISK

This lesson shows how to save programs to the disk and how to load them again. If your Apple has no disk drive, refer to Appendix B for instructions in use of tape cassettes for storage.

The commands:	SAVE	LOAD
	CATALOG	DELETE

are introduced.

If you have not yet initialized a disk for your student's exclusive use, do so before starting this lesson. Instructions and a sample HELLO program are given in an appendix.

Other commands used in this chapter are:

NEW	REM
HOME	PRINT
LIST	

This lesson can be used anytime after lesson 3.

We put it this late in the book because most programs up to this point are relatively short and uninteresting, not worth saving. The process of programming was being emphasized, not the end result of useful programs.

However, your own judgement should prevail, and you can insert this chapter at an earlier point in the flow of lessons so that your student can save some programs he/she is particularly proud of.

### QUESTIONS:

1. What is a "file"?
2. How long can a file name be?
3. What punctuation mark cannot be in a file name? Can the file name have spaces in it?
4. If you use the SAVE command without a file name, how can you get the computer back to normal?
5. What does the LOAD "filename" command do? What does the CATALOG command do?
6. How can you erase a file that you no longer want?
7. If a program is put into a file, is it still in memory?

## LESSON 14 SAVE TO THE DISK

If you do not have a disk, please turn to the Appendix B: SAVE TO TAPE.

### ENTERING A PROGRAM

If you already have a program in the computer at this moment, skip to SAVING A PROGRAM.

If not, enter:

```
NEW
10 REM ::::HI:::
20 HOME
30 PRINT "HI"
```

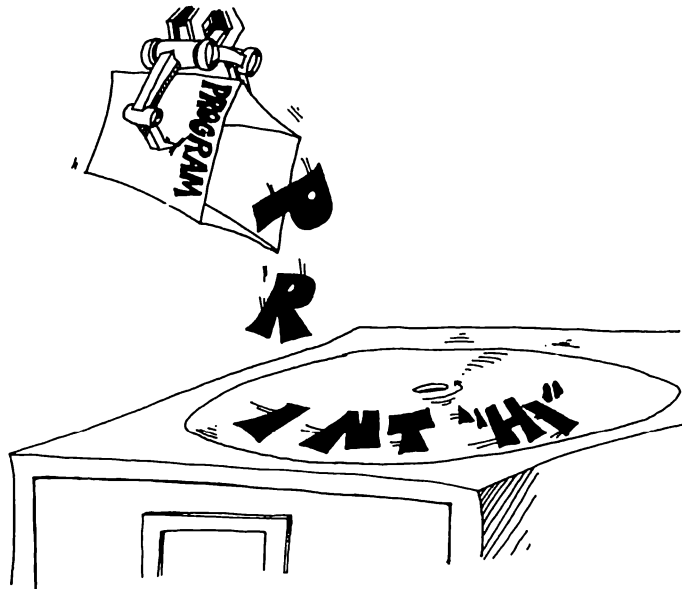
### SAVING A PROGRAM

Do you still have your disk in the drive? If not, put your disk in now. Be sure to close the door!

Enter: `SAVE HI`

You will hear a whirring and see the red light on DRIVE 1. When the red light goes off and the whirring stops, your program is stored on the disk.

The disk holds your program under the file name "HI." Think of the disk as a file cabinet. It has a file folder with the name "HI" written on it. In the file folder is your program.



We used the name "HI" because it is easier to remember if the file has the same name as the program.

If your program has a different name, SAVE it again under the correct name.

**Careful!** Do not use the word "SAVE" without a file name after it because the computer will think you want to save on a tape cassette instead of a disk.

### **A "LIFE SAVER"**

If you mess things up, press the CONTROL RESET key to get back to normal.

### **THE CATALOG COMMAND**

Let's see if the program is really stored on the disk.

Enter: CATALOG

After whirring and the red light, you will see:

A 002 HI

The "A" means the file folder contains an Applesoft program.

The 002 means it is a short program, only taking up two sectors on the disk. Think of each sector as one folder in the file cabinet.

### **LOADING THE PROGRAM**

Now that we are sure the program is on the disk, it is safe to erase it from memory.

Enter: NEW  
HOME  
LIST

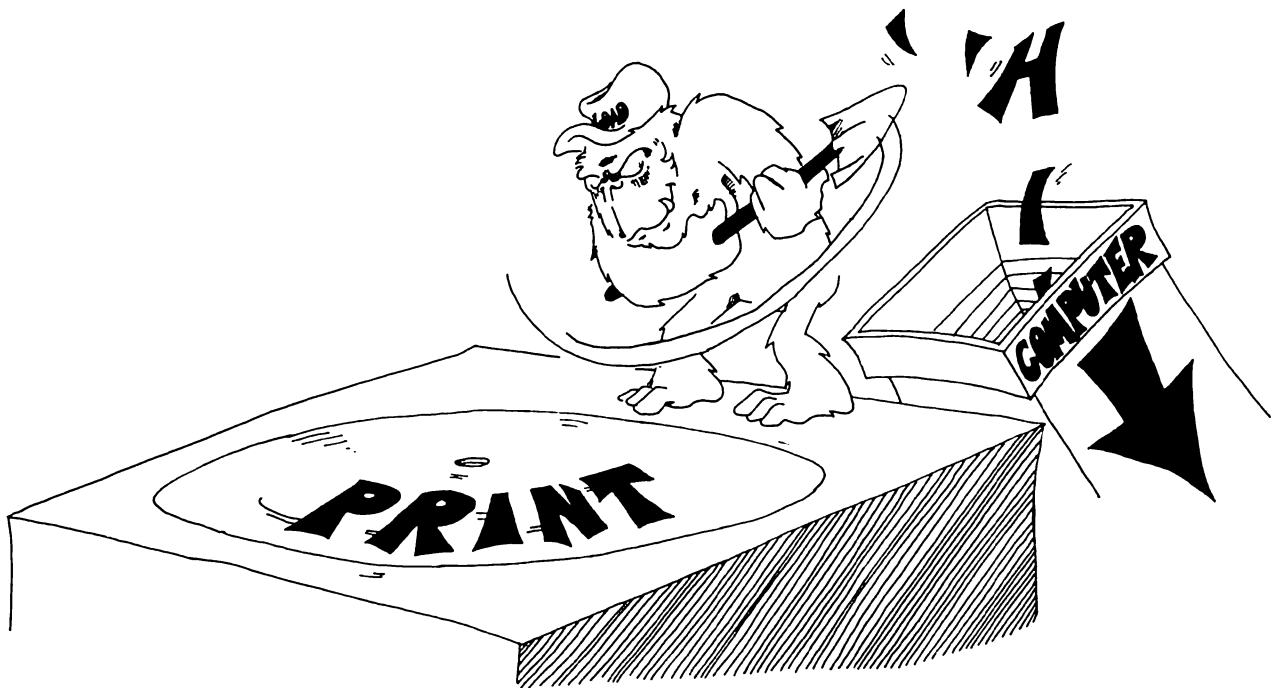
The LIST shows nothing because NEW erased the program from the computer's memory. Let's get the program back.

Enter: LOAD HI

We hear the whirring and see the red light, but is our program now in memory?

Enter: LIST

to find out.



### **ERASING A FILE**

So far, so good. But what if we change our minds and want to throw a file away?

Use `DELETE HI`

and then enter `CATALOG`

to see if it is really gone from the disk.



## LEGAL FILE NAMES

The file name:

must start with a letter  
can be long (up to 30 characters)  
can have numbers in it  
can have punctuation in it BUT . . .  
**CANNOT HAVE A COMMA IN IT**  
can even have spaces in it

A short name is best, less to type.

Good names	JUMPING CAT
	GUESSING GAME
	SUB

Wrong names	7UP	(starts with a number)
	CAT, DOG	(has a comma in it)

## COMMANDS

These four commands are used with files:

SAVE filename  
LOAD filename  
CATALOG  
DELETE filename

Where you see the word "filename" you must type the name of the file.

## TROUBLES AND ERROR MESSAGES

Always use a filename in the SAVE and LOAD commands. Otherwise the computer will think you are trying to use the tape recorder instead of the disk.

If you forget to use a filename in the DELETE command, the computer will print ?SYNTAX ERROR. Then just try again, using a file name.

### Assignment 14:

1. Write a short program (4 lines) and SAVE it on the disk.
2. Do NEW, and write another short program. SAVE it.
3. Do NEW, and CATALOG. Then load each program and run it.
4. Try out the DELETE command on one of the programs.
5. Repeat practice with the SAVE, LOAD, CATALOG, and DELETE commands until you are sure that you understand them.

# GRAPHICS, GAMES AND ALL THAT

## INSTRUCTOR NOTES 15 SOME SHORTCUTS

This lesson covers:

? used for PRINT  
LET omission  
: used between statements on a line  
INPUT used with a message

The sprint is over. We have reached RND and the saving of programs to disk. All the elements are in place for the student to write substantial programs.

The colon is used to shorten and clarify programs by putting several statements on a line. A line should contain statements that have something in common.

The colon can mess up a program too. Some statements are reached by GOTO's. If you move such a statement to the middle of another line, you will get an error message upon running the program.

A more subtle error that even experienced programmers occasionally make is to move a statement to the back of a line that has an IF in it. This changes the logic of the program, as now the statement will be executed only if the IF condition is true.

On the other hand, the colon in Applesoft allows one to put a little "subroutine" consisting of several statements after an IF. This makes using a GOTO unnecessary for reaching the extended segment of program: a shorter and much less cluttered program results. So the colon becomes a powerful and nontrivial means of improving the clarity of the program.

When INPUT is used without a message, a "?" sign is printed on the screen and the flashing input cursor appears. When INPUT is used with a message, "?" does not appear, just the flashing cursor shows. So if the message is a question, it should end in a question mark.

### QUESTIONS:

1. What shortcut does the "?" give?
2. How can you tell that the word LET is missing from a LET command?
3. An INPUT command has a message in quotation marks. What punctuation mark must follow the quotes?
4. Why is it sometimes good to put two statements on the same line, separated by a colon?
5. What is wrong with each of these lines?

```
10 REM BEGINNING:GOTO 1000  
10 GOTO 50:S$="FAST"
```

## LESSON 15 SOME SHORTCUTS

### A PRINT SHORTCUT

Instead of typing PRINT, just type a question mark.

Enter:                   10 ? "HI"  
                          LIST 10

The computer substitutes the word PRINT for the question mark.

### A LET SHORTCUT

These two lines do the same thing:

10 LET A=41 and 10 A=41

also these two:       20 LET B\$="HI" and 20 B\$="HI"

You can leave out the word LET from the LET statement! The computer knows that you mean LET whenever the line starts with a variable name followed by an "=" sign.





## AN INPUT SHORTCUT

Instead of:           10 PRINT "ENTER YOUR NAME"  
                  20 INPUT N\$

You can do:           10 INPUT "ENTER YOUR NAME"; N\$

Put a semicolon between the message "ENTER YOUR NAME" and the the variables.

Examples:            10 INPUT "AGAIN ? <Y OR N>"; Y\$  
                      20 INPUT "LOCATION"; X,Y  
                      30 INPUT "MONTH, DAY, YEAR"; M\$,D,Y

## A LIST SHORTCUT

There are 5 ways to use the LIST command:

LIST	lists whole program
LIST 48	lists line 48
LIST 50-75	lists all lines from 50 to 75
LIST -27	lists all lines from beginning to 27
LIST 90-	lists all lines from 90 to the end

## A COLON SHORTCUT

Put several statements on a line with a colon ":" between them. This saves space.

Instead of           10 Q=17\*3  
                      20 R=Q+2  
                      30 PRINT R

you can write:

10 Q=17\*3:R=Q+2:? R

In memory this line looks like:

10 Q=17\*3:R=Q+2:PRINT R

## WHEN TO USE THE COLON SHORTCUT

Use the shortcut:

1. To make the program clearer.

Put similar statements on the same line. Example:

Instead of:           10 X=0  
                  12 Y=0  
                  14 Z=0

write:                10 X=0:Y=0:Z=0

2. To make the program shorter.
3. To put a REM on the end of the line.

Example:            40 H=X+Y/66 : REM H IS THE HEIGHT

## THE COLON AFTER AN IF COMMAND

You can make neater IF statements using colons.

Without:            50 IF A=0 THEN GOTO 80  
                     60 B=0  
                     62 C=B\*D  
                     64 FLASH  
                     66 PRINT "WRONG"  
                     80 FOR . . .

With colons:        50 IF A<>0 THEN B=0:C=B\*D:FLASH:PRINT  
                      "WRONG"  
                      80 FOR . . .

All the commands in the path "A< >0 is TRUE" are on the line after THEN.

## CAREFUL!

Do not put something on the end of an IF line that doesn't belong.

Example:            35 IF A=B THEN PRINT "ALIKE"  
                     40 Q=R

is not the same as:   37 IF A=B THEN PRINT "ALIKE":Q=R

because Q=R in line 40 is always done, no matter if A=B is true or not. But Q=R in line 37 is done only if A=B is true.

### **SOME MORE MISTAKES WITH COLONS**

The REM and the GOTO commands must be last on a line. Anything following them is ignored.

Correct:                    35 P=3:REM P IS THE PRICE

Wrong:                    35 REM P IS THE PRICE:P=3

Because the computer ignores everything else on a line after reading REM.

Correct:                    40 R=P+1:GOTO 88  
                              42 S=3

Wrong:                    40 R=P+1:GOTO 88:S=3

Because the computer goes to line 88 and can never come back to do the S=3 command.

### **COMMANDS, STATEMENTS AND LINES**

Commands tell the computer to do something. So far we have used these commands:

HOME, PRINT, NEW, RUN, LIST, REM, INPUT, LET, GOTO,  
FLASH, INVERSE, NORMAL, IF, SPEED, SAVE, LOAD,  
CATALOG, DELETE

Commands used in numbered lines may be called "statements." Used alone, they are always called "commands."

Enter:    HOME            We say we have "entered a command."

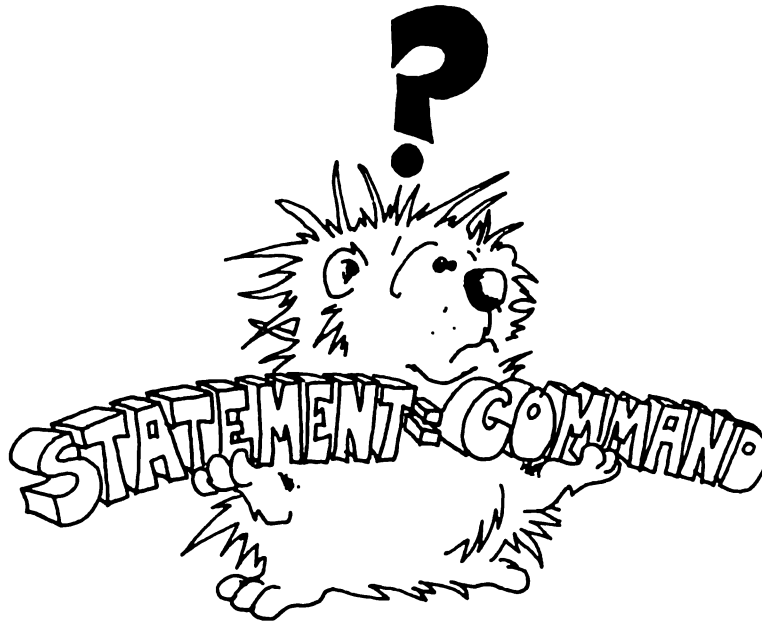
But if we write this line in a program:

20 HOME    We say that line 20 has one "statement," the HOME command.

Some lines have several statements, separated by colons.

30 HOME:PRINT:LET Z=55

is a line with three statements.



### Assignment 15:

1. Write a program that uses each of these shortcuts at least once.
2. Write a "vacation" program. It asks how much you want to spend. Then it tells where you should go or what you should do.
3. Write a "crazy" program that asks your name, then prints out a funny way of saying you are crazy. The program randomly chooses one of these and prints it after your name.

## **INSTRUCTOR NOTES 16   MOVING ABOUT ON THE SCREEN: VTAB, HTAB**

The commands HTAB and VTAB are used to move the output cursor to any point on the text screen.

These commands are used for flexible manipulation of text and/or a form of graphics.

The TAB command covered in a previous lesson is used inside a PRINT command. By contrast, HTAB and VTAB are used before the PRINT command.

To make effective use of these commands, the screen needs to be thought of as a 40 character across by 24 line down array. This viewpoint will be echoed when lo-res screen graphics are treated later.

### **QUESTIONS:**

1. If you want to print the next word on line 12, what command do you use?
2. If you want to print the next character on line 6, indented by 20 spaces, what two commands (separated by a colon) do you use?
3. In the answer to question 2, does it matter which command comes first in the line?
4. Show how to print the two words "FAT" and "CAT" on the same line with "CAT" printed first, starting at space 25, and then "FAT" printed starting at 5.

## LESSON 16 MOVING ABOUT ON THE SCREEN: VTAB, HTAB

### THE VTAB COMMAND

There is room for 24 lines of typing on the screen. VTAB chooses which line. Use any number from 1 to 24.

Run this program:

```
10 REM VTAB DEMO
15 HOME:SPEED=1
18 VTAB 10 :PRINT "LINE 10 FIRST"
20 VTAB 1 :PRINT "LINE 1 NEXT"
30 VTAB 24 :PRINT "LINE 24 LAST"
40 SPEED=255
```

And this program:

```
10 REM ::: WHICH LINE :::
20 HOME
30 INPUT "WHICH LINE";L
40 VTAB L
50 PRINT L;" HERE. ";
60 GO TO 30
```

If the number after VTAB is zero or is larger than 24, the computer prints "ILLEGAL QUANTITY ERROR IN LINE 40."

### THE HTAB COMMAND

The HTAB command tells how far over to start printing. Use numbers from 1 to 40.

It is like the TAB command you learned earlier. The TAB command is used inside the PRINT command, but the HTAB command is used by itself.

Try this:

```
10 REM --- HTAB DEMO ---
12 HOME:SPEED= 1
20 VTAB 12: HTAB 35:PRINT"HERE "
30 VTAB 12: HTAB 5:PRINT"THEN HERE"
40 SPEED=255
```

And this:

```
10 REM === WHICH ROW , COLUMN ===
20 HOME
30 VTAB 1:HTAB 1:INPUT "WHICH ROW , COLUMN?"
  " ;R,C
35 VTAB 1: HTAB 1:PRINT"  "
40 VTAB R: HTAB C:PRINT "*"
45 FOR T=1 TO 2000:NEXT T
60 GOTO 30
```

Press RESET to stop.

Line 30 wants you to type two numbers separated by commas. If you only type one number and press the RETURN key, the computer prints:

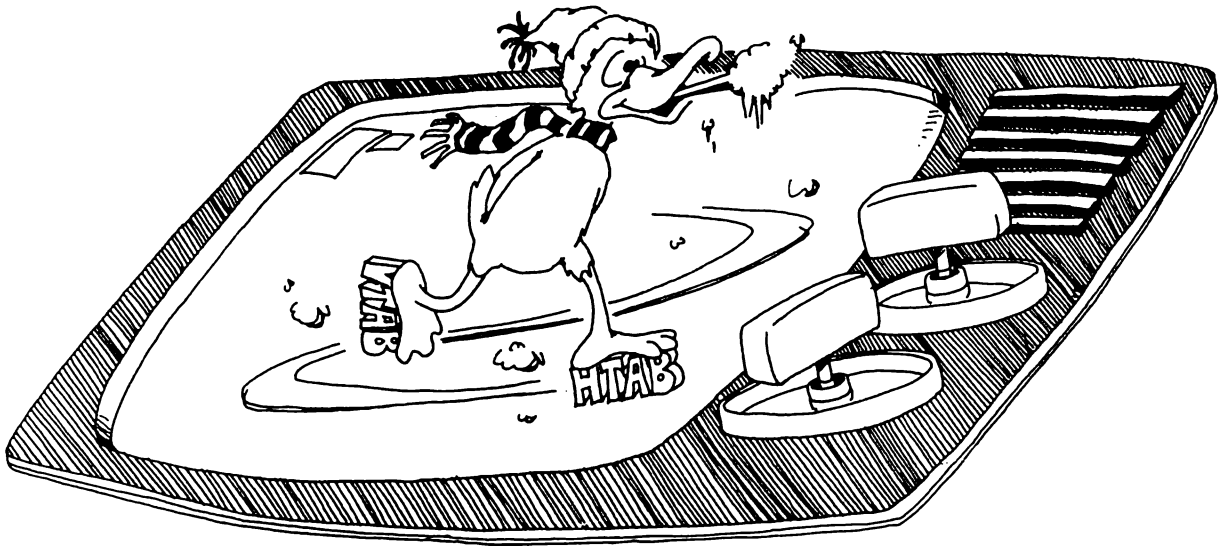
??

to tell you to enter another number.

If you type in too many numbers separated by commas, the computer prints:

**EXTRA IGNORED**

which means the extra number is not used at all.



## GRAPHS ON THE SCREEN

The screen is like a piece of graph paper. The numbers start at the top left corner.

The lines downward are numbered 1 to 24 and the variable name Y is used for them.

The columns across are numbered 1 to 40 and the variable name X is used for them. Run this program:

```
10 REM --- GRAPHS ---
20 HOME
25 PRINT "GIVE ME SOME X AND Y'S, "
27 PRINT " X FROM 1 TO 40, Y FROM 1 TO 24"
28 PRINT
30 INPUT "ENTER 'C' TO CONTINUE "; X$
32 HOME
35 INPUT X, Y
40 HTAB X : VTAB Y
50 PRINT "*"
55 HTAB 1 : VTAB 1 : PRINT "
60 HTAB 1 : VTAB 1 : GOTO 35
```

## Assignment 16:

1. Use the RND( ) function to write your first name at random points on the screen. Make it print your name many times all over the screen.
2. Write a program that prints "HERE" at random points on the screen. After each time the word is written, erase it and go on to write again. Put in a delay loop so that "HERE" does not jump around too fast.

## **INSTRUCTOR NOTES 17 FOR-NEXT LOOPS**

FOR, NEXT and STEP commands which make loops are described in this lesson.

The loop is made of two statements, one starting with FOR and the other with NEXT. These commands may be separated by several lines and yet are strongly interdependent. This could be a bit confusing to your student. The delay loop in a previous lesson helps form the notion that the FOR . . . and the NEXT are coupled. It remains then to show the utility of repeating a set of commands in the middle of the loop.

Nested loops are introduced using a case where the inside loop is a delay loop.

There are subtle points not discussed in this lesson that may arise sooner or later. The loop is always traversed at least once because the test for exit is made at the NEXT statement which can be reached only by going through the loop.

The FOR statement is evaluated just once at the time the loop is entered. It puts the starting value of the loop variable into variable storage where it is treated just as any other numerical variable. The STEP value, the ending value, and the address of the first statement after the FOR are put on a stack.

From then on, all the looping action takes place at the NEXT command. Upon reaching NEXT, the loop variable is incremented by the value of the STEP and compared with the end value. If the loop variable is larger than the end value (or smaller in the case of negative STEPS) NEXT passes control to the statement after itself. Otherwise, it sends control to the statement after the FOR command.

Because the loop variable is treated just like any other variable by BASIC, it can be used or changed in the body of the loop. Changing it should be done with care, as it will be further changed by the NEXT which also uses it to decide if the loop has ended.

Jumping into the middle of a loop is usually a disaster. Jumping out of a loop before NEXT causes an exit is commonly done, but in some cases (especially where subroutines are involved) may give hard to find bugs.

### **QUESTIONS:**

1. Write a loop that prints the numbers from 0 to 20.
2. Write a program loop that prints the numbers from 30 down to 20, by twos.
3. Write a pair of nested loops to print the numbers 100, 200, 300 and between them, the numbers 1, 2, 3, 4, 5 on separate lines.



## LESSON 17 FOR-NEXT LOOPS

Remember the delay loop? The computer counted from 1 to 2000 and then went on.

```
30 FOR T=1 TO 5:NEXT T
```

The computer is smarter than that. It can do other things while it is counting.

Run this:

```
10 REM COUNTING
20 HOME
30 FOR I=5 TO 20
40 PRINT I
50 NEXT I
```

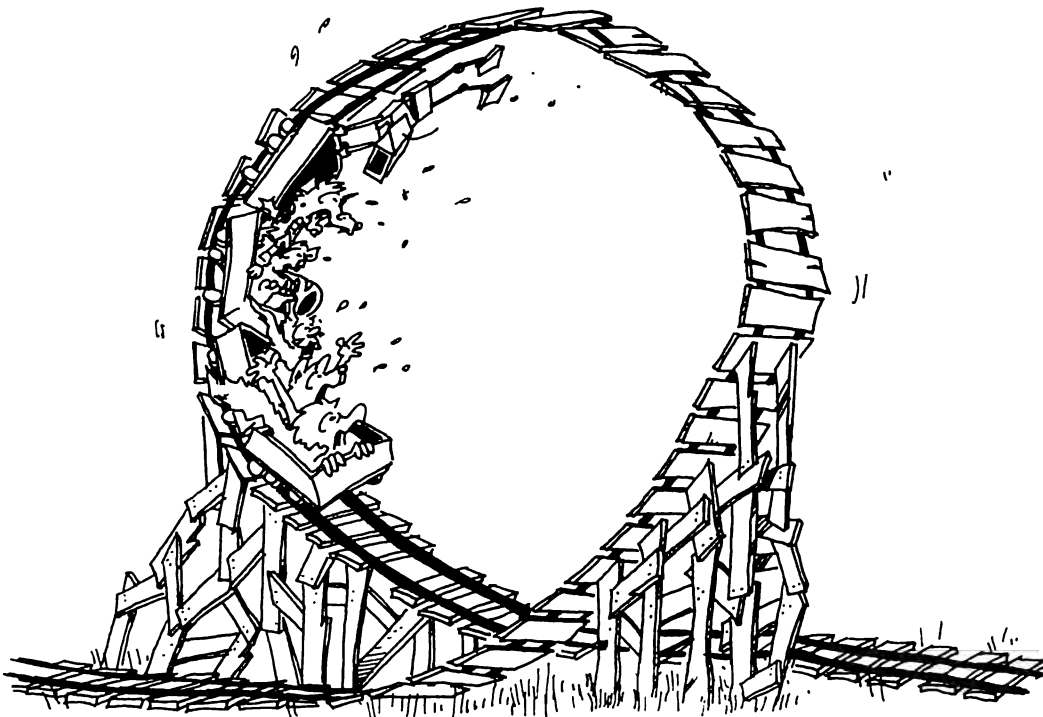

The loop can start on any number and end on any higher number. Try changing line 30 in these ways:

```
30 FOR I=100 TO 101
30 FOR I=-7 TO 13
30 FOR I=1.3 TO 5.7
```

### MARK UP YOUR LISTINGS

Show where the loops are by arrows:

```
10 REM ON PAPER
20 HOME
30 FOR I=0 TO 7
40 PRINT I
50 NEXT I
```



## THE STEP COMMAND

The computer was counting by one's in the above programs. To make it count by two's, change line 30 to this:

```
30 FOR I=10 TO 30 STEP 2
```

## Assignment 17A:

1. Have the computer count by five's from zero to 100.

## COUNT DOWN LOOPS

You can make the computer count down by using a negative STEP.

Try this:

```
10 REM *** APOLLO 11 ***
20 HOME:SPEED=100
30 PRINT "T MINUS 12 SECONDS AND COUNTING"
40 FOR I=11 TO 0 STEP -1
50 PRINT I:PRINT CHR$(7)
60 FOR J=1 TO 740:NEXT J:REM TIMING LOOP
70 NEXT I
75 INVERSE
80 PRINT "ALL ENGINES RUNNING. LIFT OFF."
82 PRINT "WE HAVE A LIFT OFF."
84 PRINT "32 MINUTES PAST THE HOUR."
86 PRINT "LIFT OFF ON APOLLO 11."
99 SPEED=255: NORMAL
```

Line 60 is the timing loop.

## NESTED LOOPS

In this program, we have one loop inside another.

The outside loop starts in line 40 and ends in line 70.

The inside loop is in line 60.

These are "nested loops." It is like the baby's set of toy boxes which fit inside each other.



## LOOP VARIABLES

To make sure that each FOR command knows which NEXT command belongs to it, the NEXT command ends in the “loop variable” name. Look at line 60:

```
60 FOR J=1 TO 740:NEXT J
```

J is the loop variable. And for the loop starting in line 40:

```
40 FOR I=12 TO 0 STEP -1
    . . .
70 NEXT I
```

I is the loop variable.

## BADLY NESTED LOOPS

The inside loop must be all the way inside:

Right:

```

┌ 25 FOR X=3 TO 7
│ ┌ 30 FOR Y=3 TO 7
│ │ 40 PRINT X*Y
│ └ 50 NEXT Y
└ 60 NEXT X
```

Wrong:

```

┌ 25 FOR X=3 TO 7
│ ┌ 30 FOR Y=3 TO 7
│ │ 40 PRINT X*Y
│ └ 50 NEXT X
└ 60 NEXT Y
```

### **Assignment 17B:**

1. Write a program that prints your name 15 times.
2. Now make it indent each time by 2 spaces more. It will go diagonally down the screen. Use TAB in a loop.
3. Now make it write your name 24 times, starting at the bottom of the screen and going up. Use VTAB in a loop.
4. Now make it write your name on one line, your friend's name on the next and keep switching until each name is written 5 times.

## **INSTRUCTOR NOTES 18   EDIT AND RUN MODES, THE CALCULATOR**

This lesson explains the EDIT MODE and the RUN MODE of the computer.

We placed this material rather late in the book, despite its fundamental nature, because it is abstract and because we did not wish to slow down the race to mastery of the core commands in BASIC.

However, you may want to take up this chapter at some earlier time in the course. The only commands used in this lesson are:

### **PRINT and RUN**

Other names for these modes are:

Edit mode:	command mode	immediate mode
	calculator mode	direct mode

Run mode:	deferred execution mode
-----------	-------------------------

The edit mode is the home base of the computer user. In the edit mode, you enter a line. The characters go into the input buffer which is 256 characters long.

When RETURN is pressed, the computer looks to see if the line starts with a number. If so, it stores the line in the program space, making room at the right location so that the lines are numbered in order.

If the line doesn't start with a number, the computer executes the line right out of the input buffer. Most commonly, the line consists of a single command, like HOME, or RUN. But the immediate mode is a very powerful one in that fairly long one line programs can be executed. This feature is handy both in the program design phase, where arithmetic concerning the design can be done in between entering lines of the program, and during debugging.

### **QUESTIONS:**

1. What does the computer do in the "RUN mode"?
2. How can you tell if the computer is in the "edit mode"?
3. What 3 kinds of things can you do in the edit mode?
4. If you enter a line that starts with a line number, what happens to the line?
5. If you enter a line that does not have a line number, what happens?

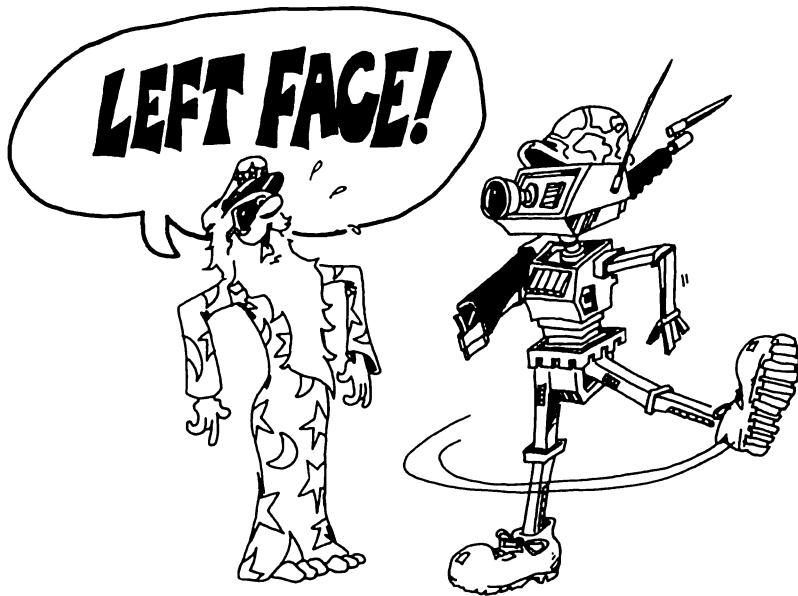
## LESSON 18 EDIT AND RUN MODES, THE CALCULATOR

Enter:                      NEW  
                                HOME      and you are ready to begin the lesson.

### EXECUTION AND RUNNING

We mean “execution” like the soldier executing the command “Left Face,” not “execution by firing squad.”

“Execute a program” means the same as “run a program.”



### DEFERRED EXECUTION

Enter and run this program:

```
10 PRINT "HI"
```

This is the usual way to make and run programs, and is called “deferred execution.”

In “deferred execution” the computer waits until you enter the command “RUN” before executing the program.

**Rule for Deferred Execution:** If the line starts with a number, it is put in memory. The line becomes part of the program in the computer’s memory. The program is executed by the command “RUN.”

### IMMEDIATE EXECUTION

Here is a short cut. Enter this (no line number in front):

```
PRINT "HI"
```

This time the computer printed “HI” right away, without waiting for you to enter RUN. This is called “immediate execution.”

**Rule for Immediate Execution:** If the line does not start with a number, the computer executes the command right away (as soon as you press the RETURN key).

Try this longer example:

```
FOR I=1 TO 20:PRINT I:NEXT I:PRINT:PRINT"DONE"
```

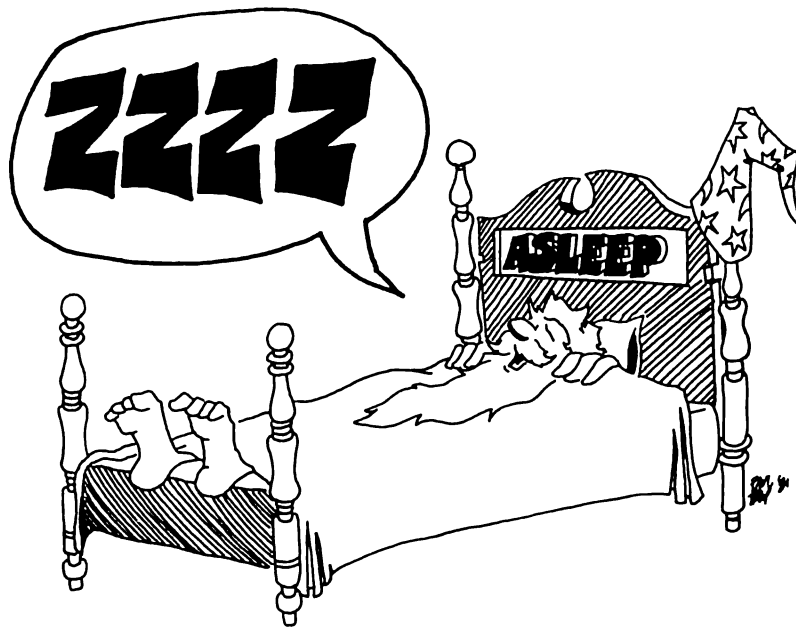
**Rule:** In immediate execution you can run a one line program that has several statements separated.

## ASLEEP OR AWAKE?

People act one way if they are awake and another way if they are asleep. They have two "operating modes."

You can tell if they are asleep because they snore. (Well, not all people snore, but to explain how computers are like people, let's pretend that all sleeping people snore.)

The computer has two operating modes too. They are called the "edit mode" and the "RUN mode."



## THE EDIT MODE

Press the CONTROL RESET key.

You see a "I" symbol and a flashing square. The "I" symbol is called a "prompt" and says that the computer is in the "edit mode" of Applesoft BASIC. The "I" is the "snoring" of the computer when it is in the edit mode.

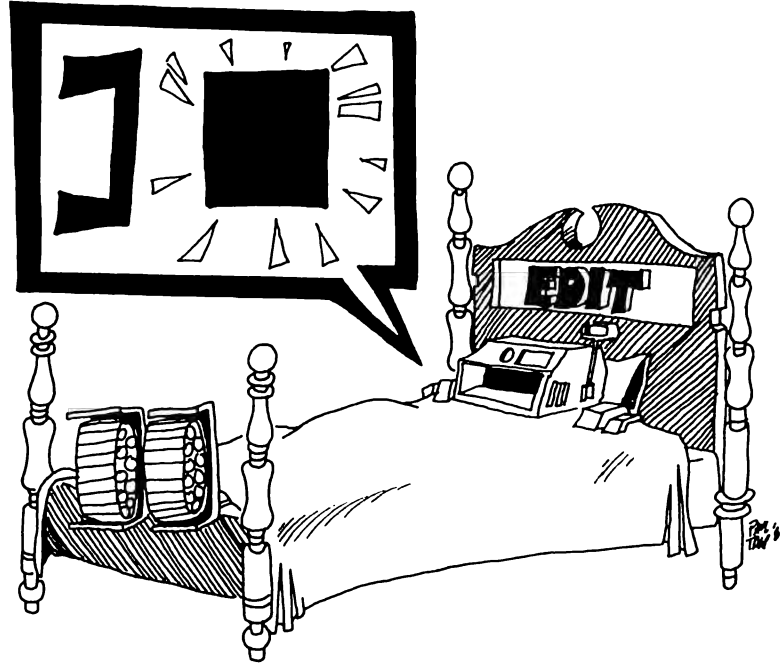
The flashing square is called the "cursor." It tells us that the computer is waiting for us to type something. The next letter we type will be on the screen underneath the flashing cursor.

While the computer is in the edit mode:

You can enter programs by typing lines that start with numbers.

You can use the computer like a pocket calculator. Big pocket!

You can correct errors in programs. This is called “editing” a program and is where the “edit mode” gets its name. Later in the book we will learn more about how to edit programs.



### THE RUN MODE

Enter RUN to leave the EDIT MODE and go to the RUN MODE.

While the computer is in the run mode:

The program in memory runs.

When the program is finished, the computer automatically goes back to the edit mode.

### Assignment 18:

1. Explain what “immediate execution” means. Use the computer as a calculator to do some arithmetic problems.
2. Explain what “deferred execution” means. Write a program that has several lines. In one line it prints “22 plus 67 is” and then in another line does the addition and prints the answer.
3. How can you tell if the computer is in the edit mode?
4. What does the computer do in the RUN mode?
5. What mode does the computer enter when the program is done running?
6. How can you tell where the next letter you type will appear on the screen?



## **INSTRUCTOR NOTES 19   MOVING PICTURES USING STRINGS**

This lesson shows an interesting little program that “shoots” an arrow across the screen.

Some of the power of string variables is demonstrated. Adding spaces moves the arrow and at the same time erases the previous image. Concatenation of strings is used and a clear “walk through” of the loop helps keep the loop idea as well as the box idea in mind.

### **QUESTIONS:**

1. For moving pictures, you must erase the old picture before drawing the new one. How is the erasing done for the “arrow” program?
2. The string box can be empty. How do you tell the computer that you want the string variable D\$ to have an empty box?
3. A loop ends with the NEXT I command. What does NEXT do to the variable I? What test does NEXT make? What choices does NEXT have as to which line is executed next?

## LESSON 19 MOVING PICTURES USING STRINGS

One way to draw pictures is to use strings. Run this:

```
10 REM >>>---ARROW---->
20 HOME
30 S$=" ":REM      S$ IS A SPACE
40 A$=">>>----->"
42 VTAB 10
44 FOR I=1 TO 29
46 HTAB I
48 PRINT S$+A$;:REM      GLUING TWO STRINGS
50 NEXT I
```

Save to disk.

This program shoots an arrow across the screen.

Line 30 There is a space between the quotation marks. So in the box named "S\$" there is stored a space.

Line 40 This string looks like an arrow.

Line 42 VTAB chooses line 10 for the arrow to move on.

Line 44 This is the start of the loop. The loop ends in line 50.

Line 46 The printing starts at the left of the line.

Line 48 First the space in box S\$ is glued on the back of the arrow. Then the space and arrow are printed.

Line 50 NEXT checks if I is larger than 29 yet. It is not. The program goes back to line 46 because this is the first line after the FOR I . . . .

Line 46 Now the printing starts one space in from the left.

Line 48 The arrow with one space on its feathered end is printed. This erases the old arrow.

Line 50 NEXT checks again.

Now you carry it two more times through the loop.

Line 46 \_\_\_\_\_

Line 48 \_\_\_\_\_

Line 50 \_\_\_\_\_

Line 46 \_\_\_\_\_

Line 48 \_\_\_\_\_

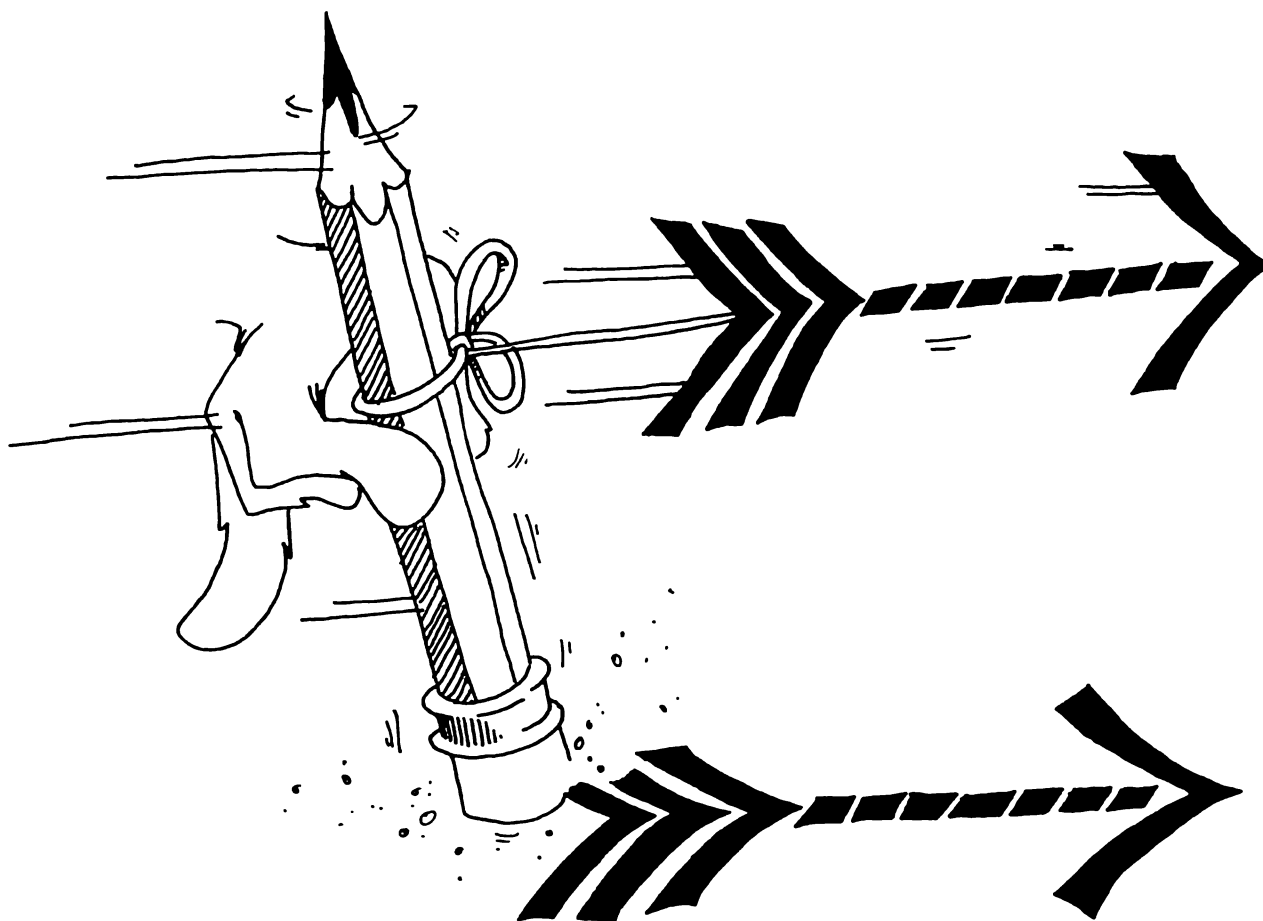
Line 50 \_\_\_\_\_

### EXACTLY HOW DOES THE ERASING HAPPEN?

Write out the printed arrow for the first three steps. Use a "b" for the blank space in back of the arrow, so you can see it. You will get:

b >>>----->  
b >>>----->  
b >>>----->

Each time, the new arrow writes over the old arrow, except for the last ">" on the feathered end. The last ">" is erased by the space on the end of the next arrow.



## EMPTY STRINGS AND STRINGS WITH SPACES

Run this program:

```
20 A$="A":B$="B"  
30 S$=" "  
40 PRINT A$;S$;B$
```

The computer prints a space between the A and the B. The space comes from line 30. It is the space between the quotes. So the box named S\$ has a space in it.

Change line 30 and run again:

```
30 S$=""
```

Now there is nothing between the quotes and so there is nothing in the box named S\$.

**Rule:** There is a difference between a string that is one space and a string that is empty.

### Assignment 19:

1. Make the arrow go slower. Make the arrow leave little puffs of smoke behind.
2. Write a program that makes your first name glide across the screen from left to right.
3. Now make another program that has your friend's name glide down from the top to the bottom. Hint: Use VTAB, and don't forget to erase the "old" name before printing the "new" name.
4. Now make a combination program where your name glides across and your friend's glides down so they cross in the middle of the screen.

## **INSTRUCTOR NOTES 20   VARIABLE NAMES**

This lesson treats the rules for naming variables.

Descriptive variable names help clarify programs. On the other hand, they take more typing and there are two hidden “gottcha’s.”

One “gottcha” is that BASIC only records the first two letters of a name. This means that those beautiful, long, descriptive names may not be unique. Even experienced programmers occasionally get caught here, and the bug may be quite hard to detect.

The other is that “reserved words” may not be included in the name anywhere: start, middle or end. The reserved words are the BASIC commands and functions.

A sign that a name contains a reserved word is provided when listing the program. The variable name will be split up showing the reserved word by itself. The first time this happens the programmer may suspect that a typing error has occurred. However, this error will pop up again after “correction” of the typing, and the only correction that works is choosing a new name.

A list of reserved words in APPLESOFT BASIC is presented in Appendix C.

### **QUESTIONS:**

1. Names longer than two characters may give you trouble. Why?
2. Names that have numbers in them may be good names or wrong names. How can you tell if they are good?
3. Names cannot have punctuation marks in them, except in one case. What punctuation mark is allowed, where do you put it, and what does it tell you?
4. Long names may have “reserved words” at the front, inside, or at the back. Which words are “reserved”? Where is there a list of reserved words?

## LESSON 20 VARIABLE NAMES

### OLD RULES

1. A string variable name ends in a dollar sign.
2. A numerical variable name doesn't.

### MORE RULES

3. A name is made of letters and numbers.
4. A name must start with a letter.
5. A name cannot have any other symbols or punctuation marks. (Except that a string name must have a "\$" at the end.)
6. A name can have as many letters and numbers as you want. But . . .
7. Only the first two characters of a name matter. All the rest are ignored (except the "\$" at the end of a string name).
8. Reserved words cannot appear anywhere in a name. These words are the ones in commands: such as REM, LIST, FOR, TO, LET, IF, PRINT, and more. Appendix B contains a list of the reserved words.

### RIGHT AND WRONG NAMES

Some correct names:    A77  
                             LEFTSIDE  
                             X3AB  
                             APE  
                             NAME\$  
                             D5\$

Some wrong names:    2X  
                             X#  
                             S[ \$  
                             \$NAME  
                             3RDNAME\$  
                             LIST  
                             COLOR                (has OR in it!)  
                             STATE\$                (has AT in it!)  
                             TOM                      (has TO in it!)

Do this. Put STATE\$ and TOM in a line like:

                             10 TOM = 1  
and then                LIST

You will see the name split apart if there is a reserved word in it.

## DIFFERENT NAMES THAT ARE THE SAME

Breaking the rule that:

“Only the first two characters matter” can make your program run very strangely!

The computer can't tell the names in these pairs apart.

It thinks that	HOSE	is the same as	HOP
and	X1	is the same as	X1Ø
and	NAME1\$	is the same as	NAME2\$

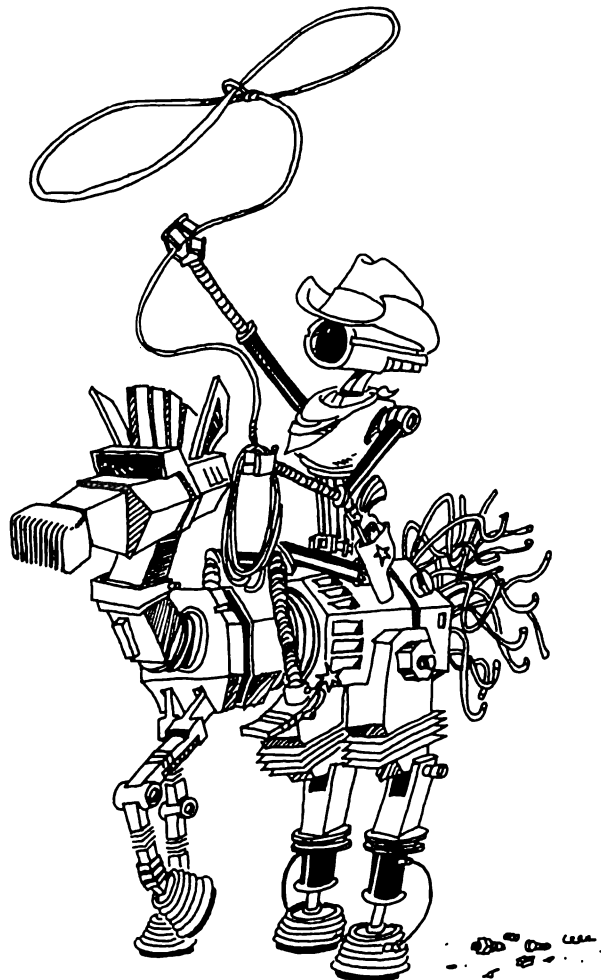
Try this program:

```
1Ø HOP$ = "BUNNY"  
2Ø HOSE$ = "LONG AND NARROW"  
3Ø PRINT HOP$
```

and see that HOSE\$ and HOP\$ name the same variable, which really just has “HO\$” written on the front of its box.

### Assignment 20:

1. Read the rules numbered 1 through 8. For each rule make up two names, one that is correct and one that disobeys the rule. (Rule number 6 has no wrong name.) Try each name in a line like 1Ø NAME=1 or 1Ø NAME\$="A" to see if it is a legal name.



## **INSTRUCTOR NOTES 21 LO-RES GRAPHICS**

This lesson introduces the commands GR, TEXT, COLOR and PLOT.

The next lesson finishes with the commands VLIN and HLIN.

If you have a black and white TV or monitor then LO-RES is still useful. In fact, the drawings will be somewhat crisper. You must pick some color other than black (number 15, white, is good).

A different style of drawing is used in color than in black and white. LO-RES color drawings look good when large areas are painted in a given color. In some colors, the lines are not very crisp.

If you have a color monitor, then the student should set up its controls for pleasing color. The COLOR DEMOSOFT program on the SYSTEM MASTER disk is good for this, which is why the appendix on preparing a disk for your student asked you to put a copy of it on the student's personal disk.

LO-RES stands for "low resolution," meaning that the spots (pixels) are not so little. They are rectangles and each is as wide as a letter in text mode of display, but only half as high. The mode that your student uses will be 40 squares wide, 40 high, and have 4 lines of text space underneath. What I call "squares" are actually rectangles.

Drawing pictures dot by dot is quite tedious. It is a little less work when using the VLIN and HLIN commands given in the next section. In any case, use of graph paper to block out the picture first is often helpful. I recommend using a variable to designate a corner (or the center) of the drawing, with offsets from the corner for the other points and lines in the drawing. Then it is easy to move the whole figure if necessary for animation or just for correction of the composition.

### **QUESTIONS:**

1. If you give the command GR what happens?
2. If you write a program using GR and PLOT, but when you run it you don't see any drawing, what have you forgotten to put after the GR command?
3. What does the TEXT command do?
4. How many colors are there to choose from?
5. What range of numbers are allowed for X and Y in the command PLOT X,Y? How is this different from the range allowed for the commands HTAB and VTAB?



## LESSON 21 LO RES GRAPHICS

Drawing pictures using little colored squares is called "LO RES graphics."

### ADJUSTING THE TV FOR COLOR

If your TV or monitor is black and white, skip to GRAPHICS COMMANDS.

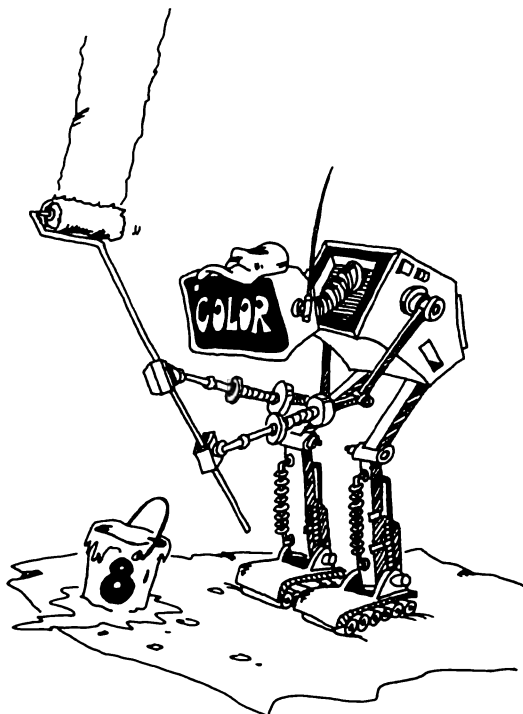
Otherwise load and run the COLOR DEMOSOFT program from your disk.

LOAD COLOR DEMOSOFT  
RUN

Choose the menu entry that gives the numbers for the colors. Turn the color and tint knobs on your color TV or color monitor so the colored stripes match the color names.

### THE COLORS ARE:

0 BLACK	8 BROWN
1 MAGENTA	9 ORANGE
2 DARK BLUE	10 GREY 2
3 PURPLE	11 PINK
4 DARK GREEN	12 LIGHT GREEN
5 GREY 1	13 YELLOW
6 MEDIUM BLUE	14 AQUAMARINE
7 LIGHT BLUE	15 WHITE



## GRAPHICS COMMANDS

```
Run:          10 REM ((( ( SMILE )))
              20 GR
              30 INPUT "WHAT COLOR? <1-15> ";C
              35 COLOR=C
              40 PLOT 20,20
              45 PLOT 21,21
              50 PLOT 22,22
              55 PLOT 23,22
              60 PLOT 24,22
              65 PLOT 25,21
              70 PLOT 26,20
              85 PRINT CHR$(7):REM PEEP
              88 HOME
              90 INPUT "AGAIN? <Y/N> ";A$
              95 IF A$="Y" THEN GOTO 30
              99 TEXT:HOME
```

SAVE SMILE

After trying some other colors, answer 0 to the "WHAT COLOR?" question. The computer will draw the smile in the color "black" and you will not see it on the black screen.

The new commands used in this program are:

GR	TEXT
COLOR	PLOT

Line 20 uses the GR command. GR means "graphics."

**Rule:** Use GR before starting your picture so that the computer knows it must draw a picture.

Line 35 uses the COLOR command. It chooses the number of the color that will be used next.

**Rule:** After using the GR command, you must use the COLOR command with a number 1 to 15.

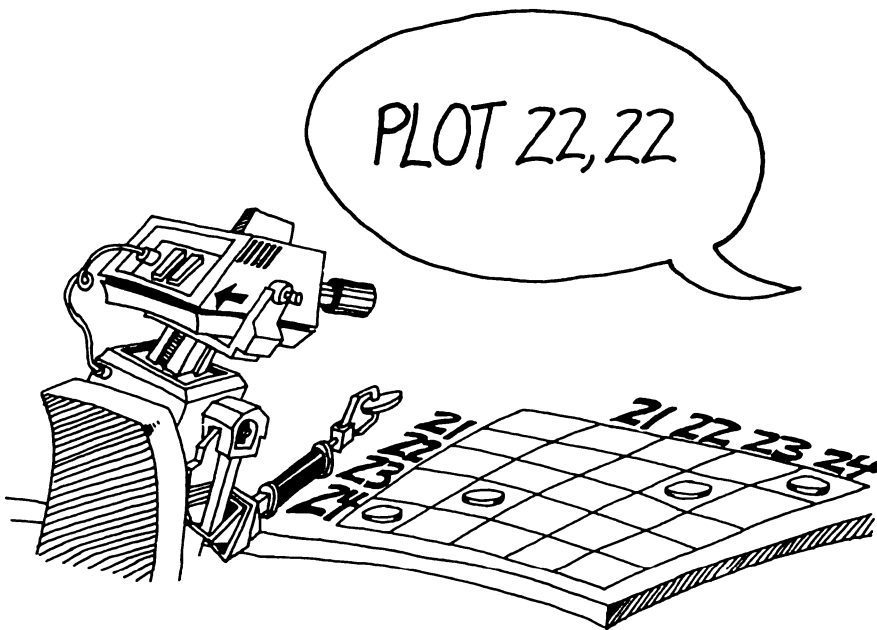
Why? Right after the GR command is executed, the color is always BLACK (which has number 0). If you forget to change to another color, you will not see a picture. You cannot see a black picture drawn on a black screen.

Line 99 uses the TEXT command.

**Rule:** End the drawing with the TEXT command so the computer will be ready to show words again.

The PLOT command is used in several lines of the program.

**Rule:** The command PLOT X,Y means put a spot on the screen at point X across and Y down. X is a number or variable in the range 0 to 39. So is Y.



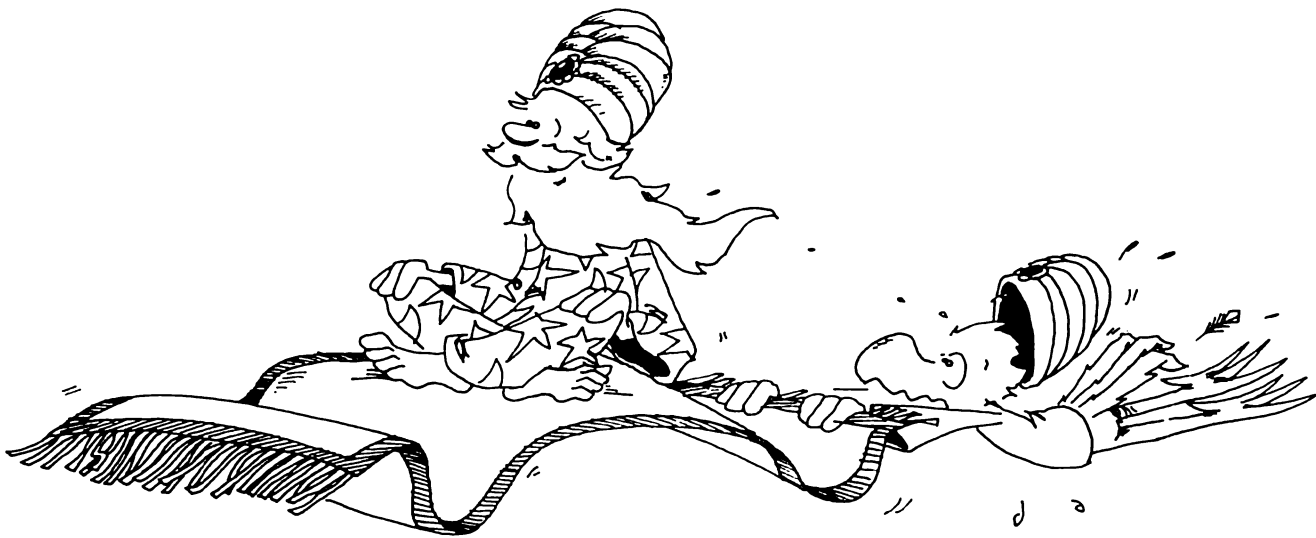
### TOO BAD!

Most vely, vely, solly. Remember the VTAB and HTAB commands go from 1 to 24 and 1 to 40 on the screen. It would be nice if the PLOT command also went from 1 to 40. But it doesn't! It goes from 0 to 39 instead.

### Assignment 21:

1. Put eyes and a nose on the face in the "smile" program.
2. Add to the number guessing game in lesson 13 so that a colored star shows when the correct answer is guessed. Use a timing loop so that the star shows for a few seconds before the game starts again.
3. Write a program to draw "Sinbad's Magic Rug." Let the user choose how many colors in the rug, and what colors. Then draw a pattern on the screen. You might like to see how the pattern was drawn in the COLOR DEMOSOFT program. Just:

```
LOAD COLOR DEMOSOFT  
LIST 700-799
```



## INSTRUCTOR NOTES 22   GRAPHICS USING HLIN AND VLIN

The line drawing commands HLIN and VLIN are explained.

LO-RES graphics use horizontal and vertical lines. Solid areas can also be filled in using lines.

### QUESTIONS:

1. In the command HLIN A,B AT C, what kind of line is drawn? What do the letters A and B tell the command? What does the letter C tell?
2. Answer the same questions for the command VLIN A,B AT C.
3. What happens if you draw one line on top of another, or crossing another?
4. How would you make a square with a blue outline and a solid red inside?



## LESSON 22 GRAPHICS USING HLIN AND VLIN

Now we will draw using lines instead of dots.

First use the COLOR DEMOSOFT program and adjust the color knobs on your TV or monitor.

### HORIZONTAL LINES

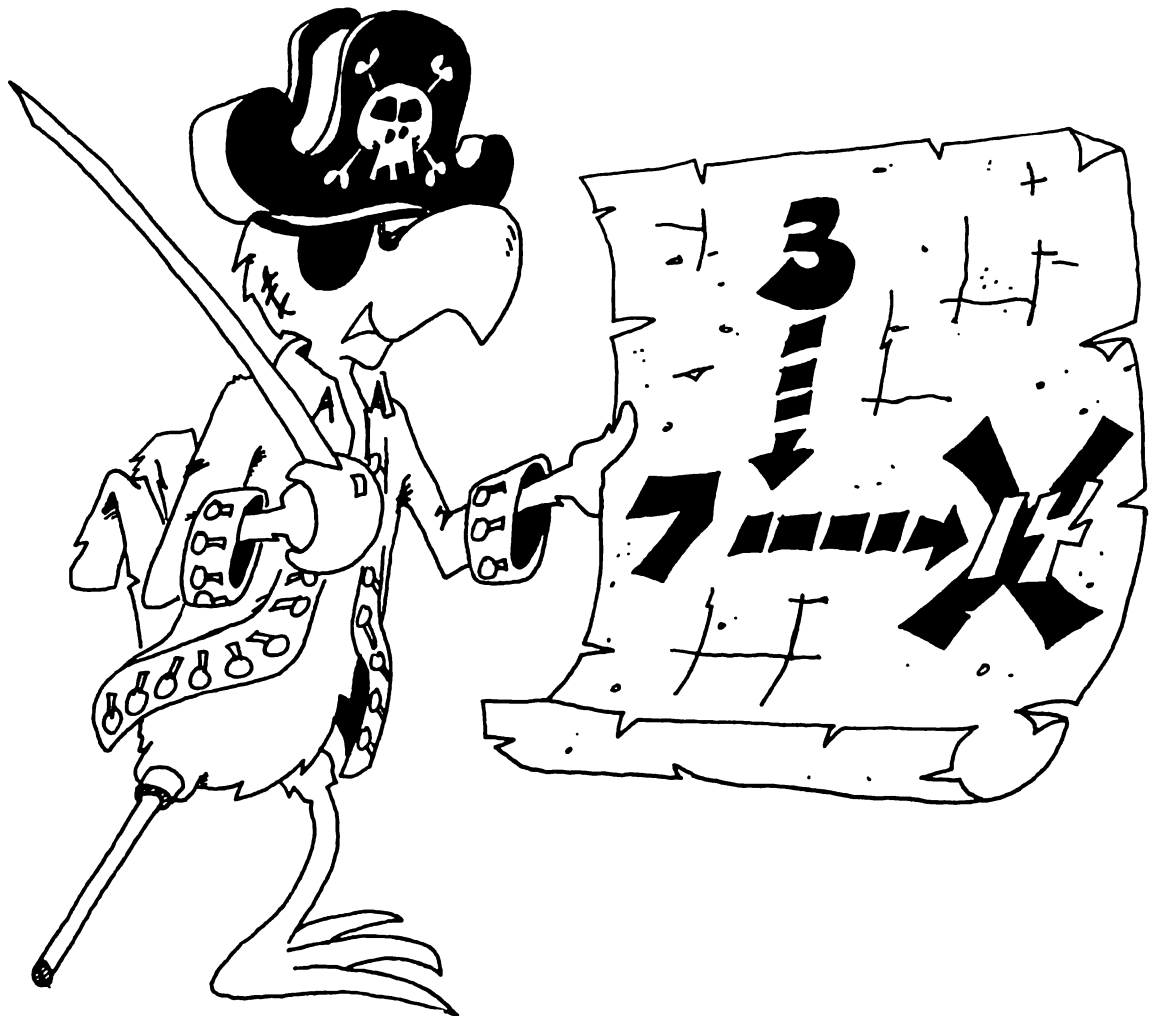
Let's draw a horizontal green line. You need to give the HLIN command with three numbers. Each number must be between 0 and 39.

```
Run:          10 GR:COLOR=12: REM COLOR IS GREEN
              30 HLIN 7,14 AT 3
```

The computer draws a green bar on the third line from the top of the screen. (That is what "AT 3" means). The green bar starts in space 7 from the left of the screen and ends in space number 14.

In other words:        30 HLIN 7,14 AT 3        means:

The line starts at 7 and goes over to 14, AT line 3 down from the top.



## VERTICAL LINES

Change line 30 to read:

```
30 VLIN 8,15 AT 4
```

Now horizontal and vertical are switched around. The command means:

The line starts at 8 and goes down to 15, AT 4 spaces in from the left

## PRACTICE PROGRAM

```
Run this: 10 GR:REM -- MAKE LINES --
          12 HOME
          15 INPUT"HORIZONTAL OR VERTICAL? <H / V> ";D$
          20 INPUT"COLOR? <1-15> ";C
          25 COLOR=C
          40 IF D$="V" THEN GOTO 70
          42 PRINT"HORIZONTAL LINE"
          44 INPUT"HOW FAR FROM THE TOP? <0-39> ";Y
          46 INPUT"START WHERE? <0-39> ";X1
          48 INPUT"END   WHERE? <0-39> ";X2
          50 HLIN X1,X2  AT Y
          55 GOTO 12
          70 PRINT"VERTICAL LINE "
          74 INPUT"HOW FAR FROM THE LEFT? <0-39> ";X
          76 INPUT"START WHERE? <0-39> ";Y1
          78 INPUT"END   WHERE? <0-39> ";Y2
          80 VLIN Y1,Y2 AT X
          85 GOTO 12
```

Save the program under file name MAKE LINES.

## A LONGER EXAMPLE

```
100 REM === PICK UP STICKS ===
105 HOME
110 GR
120 FOR I = 1 TO 50
125 FOR T = 1 TO 300: NEXT T
130 REM VERTICAL LINES
135 COLOR= RND (8) * 16
140 X = RND (8) * 39
142 Y1 = RND (8) * 39
144 Y2 = RND (8) * 39
146 IF Y1 > Y2 THEN YT = Y1:Y1 = Y2:Y2 = YT
148 VLIN Y1,Y2 AT X
155 COLOR= RND (8) * 16
157 FOR T = 1 TO 300: NEXT T
160 REM HORIZONTAL LINES
162 Y = RND (8) * 39
164 X1 = RND (8) * 39
166 X2 = RND (8) * 39
168 IF X1 > X2 THEN YT = X1:X1 = X2:X2 = YT
170 HLIN X1,X2 AT Y
190 NEXT I
192 PRINT " D O N E"
195 FOR T = 1 TO 5000: NEXT T
199 TEXT : HOME
```

Find each of these things in the program:

1. Three timing loops. What does each do?
2. One major loop.
3. Two major activities in the loop.
4. A line that returns the screen to normal when the program is done.
5. A command that tells the computer to get ready to draw.
6. How many colors are used in the drawing? Are any of the sticks black? In what lines are the colors chosen?



Enter and run the program. Then save it under the file name PICK UP STICKS.

### **MOVING YOUR PICTURE**

To make your picture move, you have to:

draw it  
erase it  
draw it again, moved over  
erase it again  
and so forth

The best way to do this is to use a subroutine to draw the picture. We will show you how to do this in lesson 24.

### **Assignment 22:**

1. Write a program that draws a square. Let the user choose what color it will be. Save it to disk.
2. Add to the program so that it has 1 to 6 spots on it like dice.
3. Write a program that draws your initials. Have it start with red (color 1) and peep, then change to dark blue (color 2) and so forth to color 15.



## **INSTRUCTOR NOTES 23   SECRET WRITING AND THE GET COMMAND**

This lesson concerns the GET command.

GET is a method of requesting a single character from the keyboard. The computer waits until the keystroke is made.

There is no screen display at all. No prompt or cursor is displayed while waiting, and the keystroke, when made, is not echoed to the screen.

The utility of the GET command lies just in this fact. For example, a requested word may be received with a series of GET's without displaying it to bystanders.

Another advantage over INPUT is that no RETURN keypressing is required. This makes GET useful in "user friendly" programming.

Apple points out that it is best to put the character into a string variable rather than a numerical variable. If you want to GET numerical digits, GET them as strings and convert them to numbers using the VAL( ) function discussed in a later lesson.

### **QUESTIONS:**

1. Compare INPUT and GET. One gets one letter at a time, the other gets whole words and sentences. One has a cursor, the other does not. One prints on the screen, the other does not. One needs the RETURN key, the other does not. Which command does which?

## LESSON 23 SECRET WRITING AND THE GET COMMAND

### THE INPUT COMMAND

There are two ways to use INPUT.

Without a message:   10 INPUT A\$  
                          10 INPUT N  
                          10 INPUT NAME\$,AGE,DAY,MONTH\$,YEAR

With a message:       10 INPUT "NAME,AGE";NAME\$,AGE  
                          10 INPUT "HOW ARE YOU? ";FEEL\$

Either way, a word, sentence or number can be typed in.

### THE GET COMMAND AND SECRET WRITING

The GET command is different from INPUT. It gets a single character from the keyboard. After the program commands GET, the computer waits until a key is pressed.

Nothing shows on the screen:

- no message will show on the screen
- no question mark will show
- no cursor will show
- what you type will not show.

The computer waits until you press one key. You do not need to press the RETURN key afterward. The computer immediately goes on with the program.



Use GET in guessing games for entering the word or number to be guessed without the other player being able to see it.

Run this program:

```
10 REM -----GET-----
20 HOME
30 PRINT "PRESS ANY KEY"
40 GET K$
45 PRINT CHR$(7)
47 FOR T=1 TO 1000:NEXT T
50 PRINT "THE KEY YOU PRESSED WAS ";K$
```

Run this one too:

```
10 REM *** BACKWARDS ***
20 HOME
30 PRINT"TYPE IN A 5 LETTER WORD"
35 PRINT
40 GET A$:GET B$:GET C$:GET D$:GET E$
50 PRINT"NOW HERE IT IS BACKWARDS"
55 PRINT:INVERSE
60 PRINT E$;D$;C$;B$;A$
70 NORMAL
```

### MAKING WORDS OUT OF LETTERS

The GET command gets one letter at a time. To make words, glue the strings.

```
10 REM GET A WORD
20 HOME
30 PRINT"TYPE A WORD. END IT WITH A PERIOD."
35 W$="": REM WORD STRING IS EMPTY
40 GET L$:REM GET A LETTER
50 IF L$="," THEN GOTO 80:REM TO TEST FOR END
60 W$=W$+L$:REM ADD LETTER TO END OF WORD
65 GOTO 40: REM TO GET ANOTHER LETTER
80 REM WORD IS FINISHED
85 PRINT W$
```

How does the computer know when the word is all typed in? It sees a period at the end of the word. Line 50 tests for the period and ends the word when it finds the period.

## THE GET COMMAND FOR NUMBERS

The GET command can be used to input numbers, but sometimes troubles appear. We will explain all this in the lesson on switching numbers with strings.

### Assignment 23:

1. Write a program that has a "menu" for the user to choose from. The user makes a choice by typing a single letter. Use GET to get the letter. Example:

```
PRINT "WHICH COLOR?  R=RED , B=BLUE ,  
G=GREEN "
```

2. Write a sentence making game. Each sentence has a noun subject, a verb, and an object. The first player types a noun (like "The donkey"). The second player types a verb (like "sings"). The third player types another noun (like "the toothpick."). Use GET so no player can see the words of the others. You may expand the game by having adjectives before the nouns.



## **INSTRUCTOR NOTES 24    PRETTY PROGRAMS, GOSUB, RETURN, END**

This lesson covers subroutines. The END command is also treated here because the program will usually have its subroutines at high line numbers and so must END in the middle line numbers.

Subroutines are useful not only in long programs but in short ones where “chunking” the task into sections leads to clarity.

One of the hardest habits to form in some students (and even some professionals) is to impose structure on the program. Structuring has gone by many names such as “structured programming” and “top down programming” and uses various techniques to discipline the programmer.

Call the student's attention to ways that structuring can be done, and the advantages in clarity of thought and ease of programming that results. In this book, writing good REM statements and using modular construction in the program are the main techniques offered.

GOSUB was put in BASIC for making modules. This lesson shows modular construction by example in the outline to the “cootie” program.

### **QUESTIONS:**

1. What happens when the command END is executed?
2. How is GOSUB different from GOTO?
3. What happens when RETURN is executed?
4. If RETURN is executed before GOSUB, what happens?
5. What does “call the subroutine” mean?
6. How many end commands are you allowed to put in one program?
7. Why do you want to have subroutines in your programs?

## LESSON 24 PRETTY PROGRAMS, GOSUB, RETURN, END

Run this program then save it to disk:

```
100 REM MAIN PROGRAM
101 :
105 HOME
110 PRINT "READY TO GO TO THE SUBROUTINE"
120 GOSUB 200
130 PRINT "BACK FROM THE SUBROUTINE"
133 PRINT
135 PRINT "GO TO THE SUBROUTINE AGAIN"
140 GOSUB 200
150 PRINT "BACK AGAIN"
190 END
199 :
200 REM SUBROUTINE
201 :
210 PRINT "IN THE SUBROUTINE"
215 FOR T = 1 TO 2000:NEXT T
217 PRINT CHR$(7)
290 RETURN
```

This is the skeleton of a long program. The main program starts at line 100 and ends at line 190.

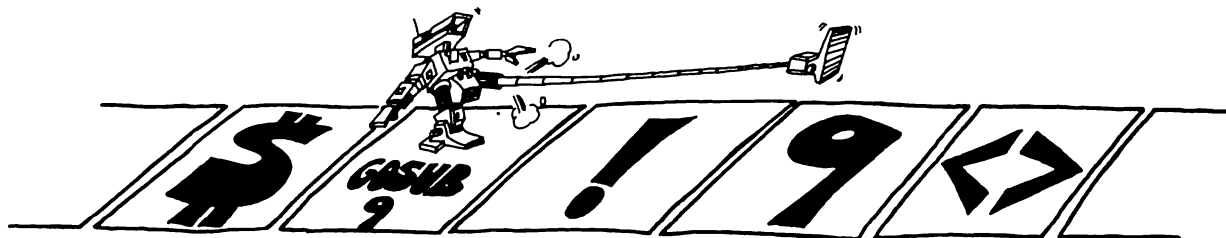
Where there are PRINT commands, you may put many more program lines.

The END command in line 190 tells the computer that the program is over. The computer goes back to the Edit Mode.

Line 120 and line 140 "call the subroutine." This means the computer does the commands in the subroutine, then comes back.

The GOSUB 200 command is like a GOTO 200 command except that the computer remembers where it came from so that it can go back there again.

The RETURN command tells the computer to go back to the statement after the GOSUB.

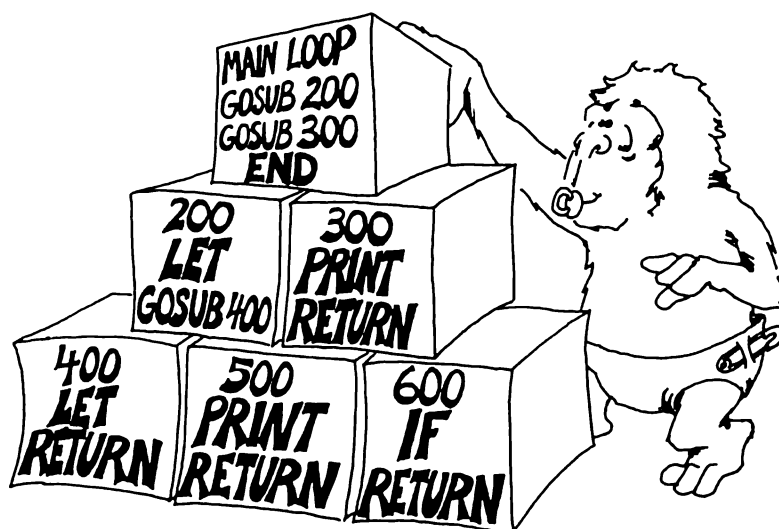


### WHAT GOOD IS A SUBROUTINE?

In a short program, not much.

In a long program, it does two things:

1. It saves you work and saves space in memory. You do not have to repeat the same program lines in different parts of the program.
2. It makes the program easier to understand and faster to write and debug.





## THE END COMMAND

The program may have zero, one, or many END commands.

**Rule:** The END command tells the computer to stop running and go back to the Edit Mode.

That is really all it does. You can put an END command anywhere in the program: for example, after THEN in an IF statement.

## MOVING PICTURES

```
10  REM ??? JUMPING J ???
20  HOME : GR
22  X = 15:Y = 15:D = 1
25  FOR J = 1 TO 10
26  FOR I = 1 TO 10
30  COLOR= 8: GOSUB 100: REM DRAW
35  COLOR= 0: GOSUB 100: REM ERASE
45  Y = Y - D
50  NEXT I
55  D = - D
60  NEXT J
90  TEXT : HOME : END
100 REM
101 REM DRAW THE J
102 REM
110 HLIN X,X + 5 AT Y
120 VLIN Y + 1,Y + 7 AT X + 3
130 HLIN X,X + 2 AT Y+ 7
140 PLOT X,Y + 6
190 RETURN
```

The picture is the letter "J". The subroutine starting in line 100 draws the "J". Before you GOSUB 100 you pick what color you want the "J" to be, using a COLOR command. Look at line 30 and at line 35. If you pick color number "0", then the subroutine erases a "J" from that spot.

The subroutine draws the "J" with its upper left corner at the spot X,Y on the screen. When you change X or Y (or both) the "J" will be drawn in a different spot. Line 22 says that the first "J" will be drawn near the middle of the screen.

The letter "D" tells how far the "J" will move from one drawing to the next. Line 22 makes "D" equal to 1, but line 55 changes D to -1 after 10 pictures have been drawn.

Line 45 says that each picture will be drawn at the spot where Y is larger than the last Y by the amount D.

## Assignment 24A:

1. Enter the JUMPING J program and run it. Then make these changes:

Change the subroutine so it prints your own initial.

Change the color of your initial to blue.

Change the “jumping” to “sliding” (so the J moves horizontally instead of vertically).

Change the starting point to the lower right-hand corner instead of the middle of the screen.

Change the distance the slide goes to 20 steps instead of 10.

Change the size of each step from 1 to 2.

Change the “sliding” so it slides uphill. Use

$$X = X + D : Y = Y - D$$

Change the program so the initial changes color from red (color 1) through all the colors to white (color 15) as it jumps.

Change COLOR = 0 in line 35 to COLOR = 1. What happens?

## HOW TO WRITE A LONG PROGRAM

Let's write a hangman game. This is a word guessing game where you draw another part of the hanging person each time you make a wrong guess for a letter.

First make an outline. You can do this on paper or right on the screen. If you have trouble deciding what to do, then just play through a game on paper and keep track of what happens. Then the program has to do the same things.

The outline could be:

```
10 REM *** HANGMAN GAME ***
200 REM INSTRUCTIONS
300 REM GET THE WORD TO GUESS
400 REM MAKE A GUESS
500 REM TEST IF RIGHT
600 REM ADD TO THE DRAWING
700 REM TEST IF GAME IS OVER
800 REM END GAME MESSAGE
```

After making this outline, I filled in more details.

```

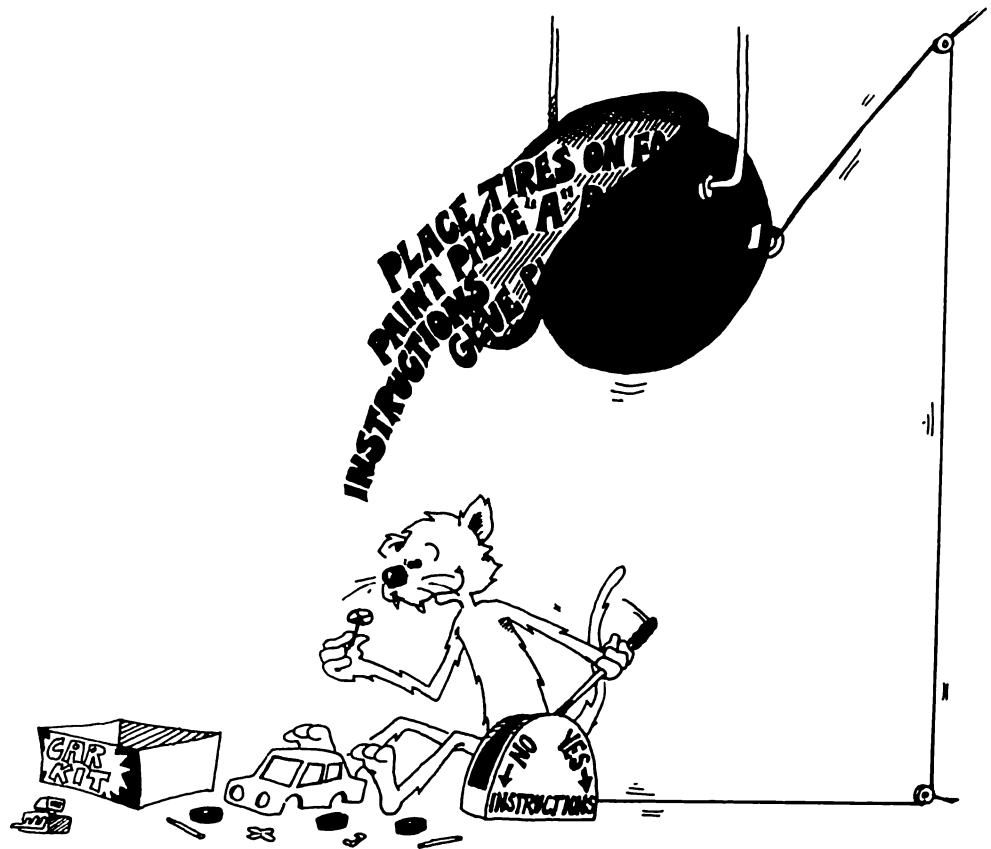
10  REM *** HANGMAN GAME ***
99  :
100 REM MAIN LOOP
101:
120 INPUT" NEED INSTRUCTIONS? <Y/N> "; Y$
122 IF Y$="Y" THEN GOSUB 200
130 GOSUB 300:REM GET WORD
132 STOP
135 GOSUB 400:REM MAKE GUESS
140 GOSUB 500:REM TEST GUESS
145 GOSUB 700:REM TEST IF GAME IS OVER
190 GOTO 135:REM MAKE ANOTHER GUESS
199:
200 REM INSTRUCTIONS
...  write the instructions last
290 RETURN
299:
300 REM GET THE WORD TO GUESS
...  use INPUT to get a word from player 1
...  draw dashes for the letters to be guessed
390 RETURN
399:
400 REM MAKE A GUESS
...  player 2 guesses a letter
490 RETURN
499:
500 REM TEST IF GUESS IS RIGHT
...  if wrong, GOSUB 600:REM draw hangman part
...  if right, GOSUB 700:REM see if game is over
590 RETURN
599:
600 REM ADD TO THE DRAWING
...  add to the hangman drawing
...  test if drawing is done
...  if so, then GOSUB 800
690 RETURN
699:
700 REM TEST IF GAME IS OVER
...  see if all letters have been guessed
...  if yes, GOSUB 900
790 RETURN
799:
800 REM END GAME MESSAGE
...  message for when guesser loses
890 RETURN
899:
900 REM END OF GAME MESSAGE
...  message for when guesser wins
990 RETURN

```

Things are getting a little mixed up in my mind on how to end the game. So I will leave that to later and start writing and testing the first part of the program. I put a STOP in line 132 so that only the first subroutine will be run. I will start by writing the subroutine at 300, GET A WORD.

### Assignment 24B:

1. Write a short program that uses subroutines. It doesn't have to do anything useful, just print some silly things. In it put three subroutines:  
Call one of them twice from the main program.  
Call one of them from another of the subroutines.  
Call one of them from an IF statement.
2. Write a program that writes your 3 initials on the screen, each one a different color. Then make them jump up and down one at a time!
3. Finish the hangman game. This is a long project, and you may want to do part of it now and SAVE it to disk and finish the game later.



## ADVANCED PROGRAMMING

### INSTRUCTOR NOTES 25 LINE EDITING

Line editing is a slightly involved procedure but nevertheless is cut and dried.

This lesson introduces the     I  
                                  J   K  
                                  M

system of moving the cursor. The ESC key must be pushed once before moving the cursor with the I, J, K, and M keys.

Once the cursor is somewhere in the line to be fixed, it is moved to the first digit of the line number using the J key. This is important. Do not use an arrow key for this move to the front. After the cursor is placed on the first digit, only the arrow keys are used to move the cursor back and forth in the line. The I, J, K, and M keys must not be used.

The reasons are this. The arrow keys also move a cursor back and forth in the line buffer. The right arrow reads characters from the screen into the buffer. If the buffer is to start with the line number, the arrow keys must not be used until the cursor is on the first digit of the line number. From then on, the line buffer contents will match the characters on the screen if the arrow keys and the typing of characters are the only cause of cursor movement.

Appendix B of the Applesoft BASIC Programming Reference Manual gives a somewhat more complete description of line editing. It includes cases on insertion and deletion in lines. These procedures are recommended only after mastering the line editing procedures given here.

#### QUESTIONS:

1. To move the cursor do you hold the ESC key down while pressing the I, J, K, and M keys?
2. To edit a line and put it back in memory, can you use the J and K keys in place of the arrow keys? Sometimes? Never? Always? Explain.
3. What happens if you use the arrow keys to put the flashing cursor in the middle of the line, then press RETURN?

## LESSON 25 LINE EDITING

We are going to use the “ESC” key. ESC stands for “escape.” We will let the flashing cursor “escape” from its line and go wandering all over the screen.

### MOVING THE CURSOR ANYWHERE ON THE SCREEN

To move the blinking cursor to any spot on the screen, follow the rule.

**Rule:** First press the ESC key once. Then press these keys as often as you want:

	I	go up
go left	J K	go right
	M	go down

(Use the REPT key too.)

**Careful!** Don’t use the arrow keys to go left or right! Don’t use any other keys except I, J, K, and M. (If you do, you must press ESC once more.)

### PRACTICE MOVING THE CURSOR

Try moving the cursor to the top of the screen.

Now move it to the left, then the right. If it goes off the edge of the screen, it pops back on the other side.

Now move it back to the bottom.

Move it up and in a circle.



## **FIXING A LINE**

You learned this before.

To fix a line that you are typing:

1. Use the arrow keys to move the cursor to the error.
2. Type the correct letter.
3. Use the arrow to move to the end of the line.
4. Press RETURN to put the line in memory.

## **LINE EDITING: CHANGING A LINE THAT IS IN MEMORY**

Sometimes you don't find the error until after the line is put in memory.

Sometimes you just change your mind about the line.

Do you have to type the line over? No!

Do these three things:

1. Use LIST to put the line on the screen.
2. Move the cursor to the beginning of the line using the ESC and the I, J, K, and M keys.
3. Fix the line using the arrow, REPT, and RETURN keys just as before.

**Careful:** Use only the ESC and I, J, K, and M keys. Put the cursor all the way to the left in the line you want to fix.

After the cursor is over the first digit of the line number, do not use the I, J, K, or M keys any more.

Use only the arrow keys to move to the location of the mistake. Type to fix the mistake.

When the line is correct, move the cursor to the end (use right arrow key) and press RETURN to put the line back in memory.

**Sorry:** You cannot add extra letters into the middle of the line. But you can blank out letters by putting a space in place of the letter.



### **PRACTICE LINE EDITING**

Enter:                    623 PRINT "I LIKE IPPLES"  
                             HOME

Now enter:              LIST 623

Press ESC once and then the "I" key a few times to get the flashing cursor on top of the "2."

Press the "J" key to move the cursor over the "6."

Press the right arrow and REPT to move the cursor over the first letter in "IPPLES."

Press the "A" key. This fixes the word to be "APPLES."

Press the right arrow key until the cursor is in the next space past the quotes at the end of the line. Press RETURN.

Then LIST to see if the line is correct.

### **Assignment 25:**

1. Load one of your old programs from disk and practice LISTing lines and using ESC and I,J,K, and M keys to reach the first digit of the line number. Then change the lines using the arrow keys to move around. Store the lines back in the program using the RETURN key.
2. Write an APPLE TREE program. It draws a background of blue, then grey ground, brown trunk and green leaves. Put apples in the tree. Then have the apples fall to the ground. Use subroutines for drawing the parts of the tree, and for making the apples fall. Use line editing to fix any bad lines.



## INSTRUCTOR NOTES 26 SNIPPING STRINGS: LEFT\$, MID\$, RIGHT\$, LEN

In this lesson the functions:

LEFT\$	MID\$	RIGHT\$
LEN		

are demonstrated. The use of MID\$() with 3 arguments is shown, but not that with the third argument omitted.

These functions together with the concatenation operation “ + ” allow complete freedom to cut up strings and glue them back in any order.

As in earlier explanations, the main characteristics of the functions are shown, but not all the special cases, especially those that lead to ERROR messages. It is better that extensive explanations not clutter up the text. If the student experiences difficulty, an experienced programmer or an adult consulting the Apple manuals should clear up the problem.

### QUESTIONS:

1. If you want to save the “STAR” from “STARS AND STRIPES,” what function will you use? What arguments?
2. If you want to save “AND”, what function and arguments?
3. If you want to count the number of characters in the string PQ\$, what function do you use? What argument?
4. What is wrong with each of these lines?

```
10 A$=LEFT$(4,D$)
10 RIGHT$(R$,I)
10 F$=MID$(A,3)
10 J$=LEFT(R$,YT)
```

5. What two arguments does the RIGHT\$( ) function need?
6. What command will snip the third and fourth letters out of a word.
7. Write a short program that takes the word “computer” and makes it into “putercom.”

## LESSON 26 SNIPPING STRINGS: LEFT\$, MID\$, RIGHT\$, LEN GLUING STRINGS

You already know how to glue strings together:

```
55 A$="CON" + "CAT" + "EN" + "ATION"  
60 PRINT A$
```

The real name for “gluing” is “concatenation.”

Concatenation means “make a chain.” Maybe we should call them “chains” instead of “strings.”

### SNIPPING STRINGS

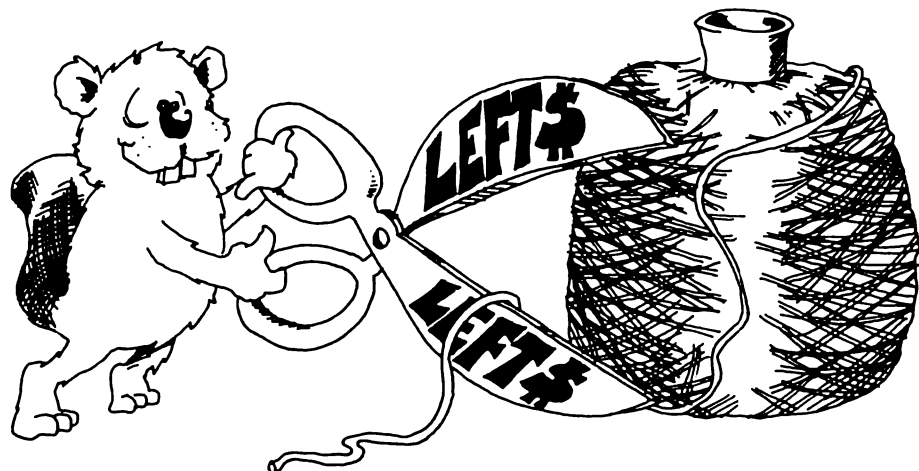
Let’s cut a piece off a string. Enter and run:

```
10 REM >>> SCISSORS >>>  
20 HOME  
30 N$="123456789"  
35 Q$=LEFT$(N$,4)  
40 PRINT Q$, N$
```

The LEFT\$ function snips off the left end of the string. The snipped off piece can be put in a box or printed or what ever.

**Rule:** The LEFT\$( ) function needs two things inside the ( ) signs.

1. The string you want to snip.
2. The number of characters you want to keep.



Try another. Change line 40 to:

```
40 PRINT RIGHT$(N$,3)
```

and run the program again. This time the computer prints:

```
789
```

RIGHT\$( ) is like LEFT\$( ) except the characters are saved off the right end of the string.

### **MORE SNIPPING AND GLUING**

```
Run:      10 REM ::: SCISSORS AND GLUE :::
          20 HOME
          30 N$="123456789ABCDEF"
          35 FOR I=1 TO 15
          40 L$=LEFT$(N$,I) : R$=RIGHT$(N$,I)
          50 PRINT I;TAB(5);L$;TAB(25);R$
          60 NEXT I
```

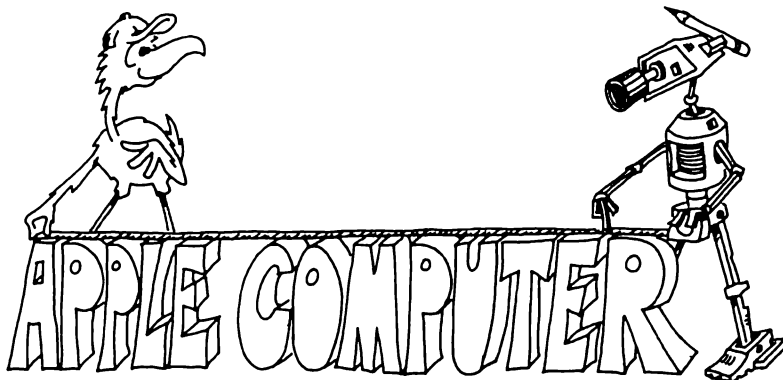
The pieces of string you snip off can be glued back together in a different order. Add this line and run:

```
55 IF I=4 THEN PRINT: PRINT R$ + L$ :PRINT
```

### **HOW LONG IS THE STRING?**

```
Run:      10 REM ::: LONG ROPE :::
          20 HOME
          30 INPUT"GIVE ME A STRING: ";N$
          40 L=LEN(N$)
          50 PRINT "THE STRING: ";N$;"'"
          55 PRINT:PRINT "IS ";L;" CHARACTERS LONG"
```

The function LEN( ) tells the number of characters in the string. It counts everything in the string, even the spaces.



## CUTTING A PIECE OUT OF THE MIDDLE

The MID\$( ) function cuts a piece out of the middle of the string.

Run:

```
10 REM *** MIDDLE ***
20 HOME
30 N$="123456789"
40 P$=MID$(N$,3,4)
50 PRINT P$
```

The line: 40 P\$= MID\$(N\$,3,4) means:

Get the string from box N\$.

Count over 3 letters and start saving letters into box P\$.

Save 4 letters.

## LOOK MA, NO SPACES

Enter:

```
10 REM >>> NO SPACES >>>
11 :
20 PRINT : PRINT
25 PRINT"GIVE ME A LONG SENTENCE": PRINT
30 INPUT S$
40 L =LEN (S$)
45 T$ = ""
50 FOR I = 1 TO L: REM LOOK AT EACH LETTER
60 L $= MID$ (S$,I,1)
70 IF L$ < > " " THEN T$ = T$ + L$: REM SAVE ONLY
LETTERS
90 NEXT I
92 PRINT : PRINT T$
95 PRINT : PRINT : PRINT
```

Line 60 snips just one letter at a time out of the middle of the string.

## Assignment 26:

1. Write a secret cipher making program. You give it a sentence and it finds how long it is. Then it switches the first letter with the second, third with the fourth, etc. Example:

THIS IS AN APPLE. becomes:

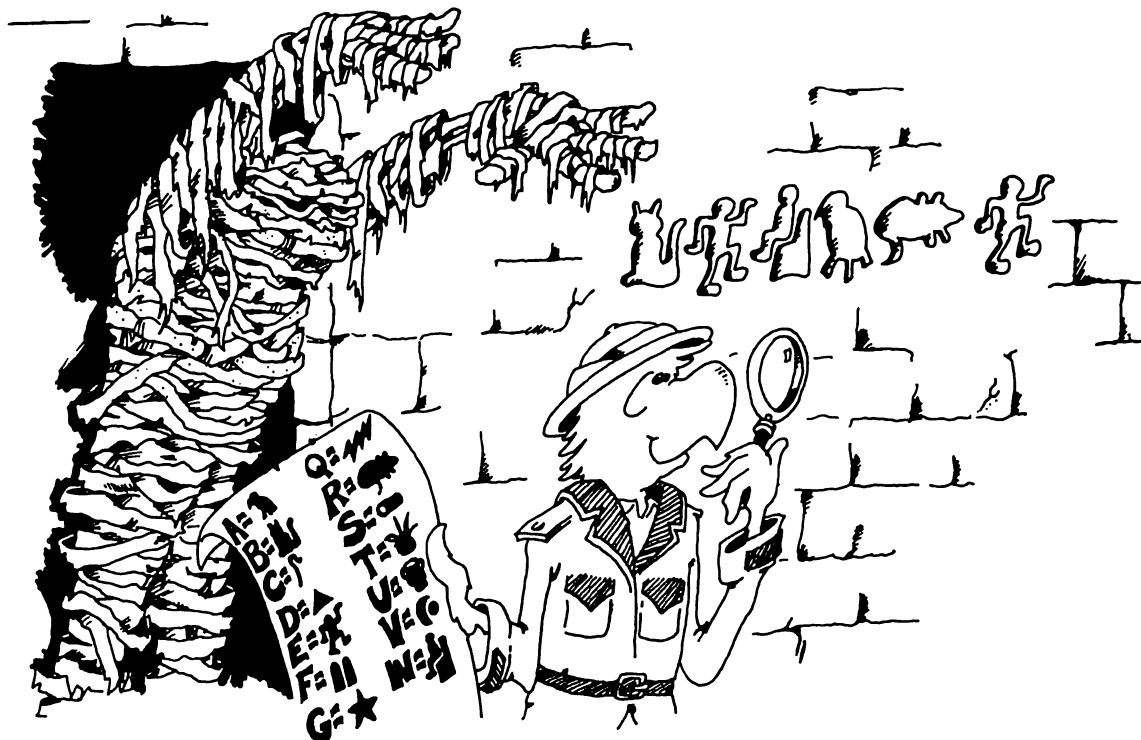
HTSI SI NA PALPE

2. Write a question answering program. You give it a question starting with a verb and it reverses verb and noun to answer the question. Example:

ARE YOU A TURKEY?

YOU ARE A TURKEY.

3. Write a PIG LATIN program. It asks for a word. Then it takes the first letter and puts it at the end of the word and adds AY. If the first letter is a vowel, it adds LAY, LEE, LI, LO, or LU.



## **INSTRUCTOR NOTES 27 SWITCHING NUMBERS WITH STRINGS**

This lesson treats two functions, STR\$ and VAL. A general review of the concept of function is also made.

STR\$ takes a number and makes a string that represents it.

VAL does just the opposite, taking a string and making a numerical value from it.

This interconversion of the two main types of variables adds great flexibility to programs involving numbers.

### **QUESTIONS:**

1. If your number “marches” too quickly in the program of assignment 27, how do you slow it down?
2. If your program has the string “GEORGE WASHINGTON WAS BORN IN 1732.” write a few lines to answer the question “How long ago was Washington born?” (You need to get the birthdate out of the string and convert it to a number.)
3. What is a “value.” What is meant by “a function returns a value?” What are some of the things you can do with the value?
4. What is an “argument” of a function? How many arguments does the RIGHT\$( ) function have? How many for the CHR\$( ) function?
5. Where in the line do commands always go? Can you put a function at the start of a line?

## LESSON 27 SWITCHING NUMBERS WITH STRINGS

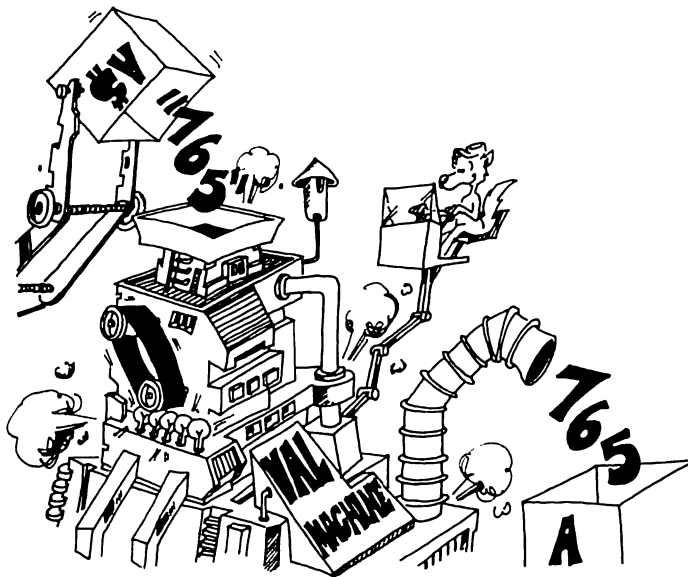
This lesson explains two functions: VAL( ) and STR\$( ).

### MAKING STRINGS INTO NUMBERS

We have two kinds of variables, strings and numbers. We can change one kind into the other.

```
Run:          10 REM MAKING STRINGS INTO NUMBERS
               20 HOME
               30 L$="123"
               40 M$="789"
               50 L=VAL(L$)
               60 M=VAL(M$)
               70 PRINT L
               72 PRINT M
               74 PRINT "---"
               76 PRINT L+M
```

VAL stands for "value." It changes what is in the string to a number, if it can.



### MAKING NUMBERS INTO STRINGS

```
Run:          10 REM MAKING NUMBERS INTO STRINGS
               11 :
               20 PRINT
               25 INPUT"GIVE ME A NUMBER ";NB
               30 N$=STR$(NB)
               35 L=LEN(N$)
               37 PRINT
               40 FOR I=L TO 1 STEP -1
               45 B$=B$+MID$(N$,I,1)
               50 NEXT I
               60 PRINT"HERE IT IS BACKWARDS"
               65 PRINT:PRINT B$
```

STR\$ stands for "string." It changes a number into a string.

## FUNCTIONS AGAIN

In this book we use these functions:

RND( ), INT( ), LEFT\$( ), RIGHT\$( ), MID\$( ), LEN( ), VAL( ),  
STR\$( ), ASC( ), CHR\$( ), SCRN( ), PDL( )

RULES about functions:

Functions always have ( ) with one or more "arguments" in them. Example:

MID\$(D\$,5,J) has 3 arguments: D\$, 5, and J

The arguments may be numbers or strings or both.

A "function" is not a "command." It cannot begin a statement.

right: 10 LET D=LEN\$(CS\$)

wrong: 10 LEN (CS\$)=5

A function acts just like a number or a string. We say the function "returns a value." The value can be put in a box or printed just like any other number or string. The function may even be an argument in another function.

The arguments tell which value is returned.

(Remember, string values go in string variable boxes, numerical values go in numerical boxes.)

## PRACTICE WITH FUNCTIONS

For each function in the list below:

Tell how many arguments it has, and give their names.

Tell whether the value of the function is a string or a number:

RND ( B ) \_\_\_\_\_  
INT ( Q ) \_\_\_\_\_  
LEFT\$ ( U\$ , Y ) \_\_\_\_\_  
MID\$ ( RIGHT\$ , E , 2 ) \_\_\_\_\_  
VAL ( ER\$ ) \_\_\_\_\_  
STR\$ ( INT ( RND ( B ) ) ) \_\_\_\_\_

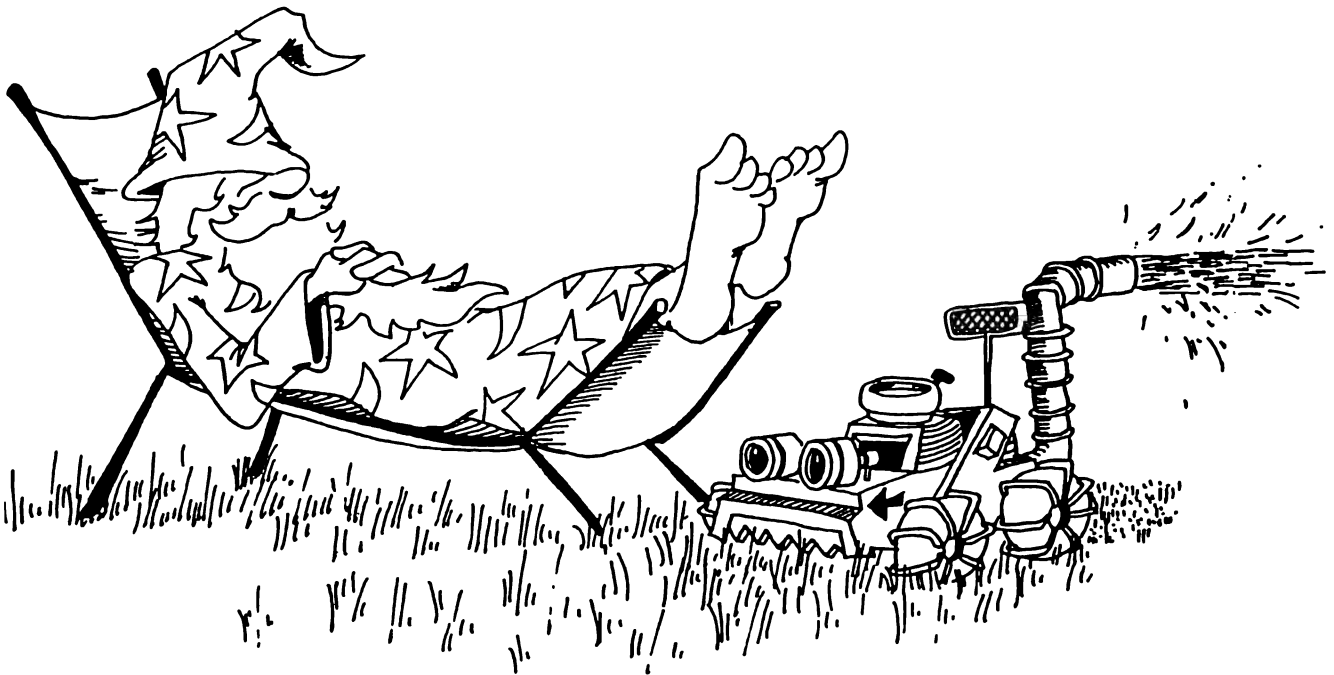


Each line below has errors. Explain what is wrong.

```
10 INT(Q)=65 _____  
10 D$=LEFT(R$,1) _____  
10 PW$=VAL(F$) _____  
10 PRINT CHR$ _____
```

### Assignment 27:

1. Write a program that asks for a number. Then make another number that is backwards from the first and add them together. Print all three numbers like an addition problem (with “+” sign and a line under the numbers).
2. Make a number “march” slowly across the screen. That is, write it on the screen, then take its left digit and move it to the right. Keep repeating. Don’t forget to erase each digit when you move it.



## INSTRUCTOR NOTES 28 PADDLES FOR ACTION GAMES

This long lesson introduces the PDL( ) and SCRN( ) functions. It also uses PEEK(-16287) and PEEK(-16286) to see if the paddle buttons have been pushed.

Paddles are commonly used in animated graphics games. In this lesson LO-RES graphics are used to move a dot (perhaps a cannon) which can shoot. The student will need to understand the X,Y addressing of the squares on the 40 by 40 screen.

When drawing moving objects, you need to erase each old image before the next image is drawn.

A "hit" on a target is detected by using SCRN( ) to test the square in front of the projectile. If there are several kinds of targets, it is better to test if the square is "background." If not, then jump to a subroutine that asks the color and thus the target type.

Graphics games may grow to be rather long. BASIC is a little slow for such games. Maximum speed can be obtained if the "working" part of the program is first, and the "initialization" part is at the end, reached by a call from early in the program.

For maximum speed, avoid repeatedly converting numbers to floating point. Such lines as 60 in the == PADDLES == program are better written as:

```
60 PEEK(Q0)
```

where a variable Q0 = -16287 is defined in the initialization section. This practice is probably the most important single factor in obtaining fast programs.

### QUESTIONS:

1. What range of numbers does the PDL( ) function return?
2. How can you change this range to be 0 to 20?
3. The PEEK(-16287) function tells if the push button on paddle 0 is being pushed. What will the value of the PEEK be if the button is being pushed? If it is not being pushed?
4. What argument does PEEK( ) have if you are asking about the push button on paddle 1?
5. What is the SCRN(X,Y) function used for? What range of numbers are allowed for X? For Y?
6. If your program makes a dot move across the screen, how can you tell if the dot is about to hit something on the screen?

## LESSON 28 PADDLES FOR ACTION GAMES

Run this program if your computer has game paddles.

### PADDLES AND THEIR BUTTONS

```
Run:      10 REM === PADDLES ===
          20 HOME:PRINT
          30 PRINT "TURN THE PADDLE KNOBS"
          35 PRINT:PRINT "AND PUSH THE BUTTONS"
          38 REM CHECK THE KNOBS
          40 P0=PDL(0)
          45 VTAB=10
          47 PRINT "PDL 0 = ";P0;TAB(15);
          48 P1=PDL(1)
          50 PRINT "PDL 1 = ";P1
          55 REM CHECK THE BUTTONS
          60 B0=PEEK(-16287)
          62 B1=PEEK(-16286)
          65 VTAB 12
          70 C$="      ":IF B0>127 THEN C$="BANG"
          72 PRINT C$;
          74 C$="      ":IF B1>127 THEN C$="BANG"
          76 PRINT TAB(15); C$
          99 GOTO 40
```

Use the RESET key to end the program. Save to disk.

### THE GAME PADDLES

There are 4 paddles, numbered 0, 1, 2, and 3. (Your computer may have only 2 paddles, numbered 0 and 1, or may not have any paddles.)

The PDL(0) function tells how much the knob on paddle 0 has been turned.

If the number is near zero, the knob has been turned to the left.

If the number is near 255, the knob has been turned to the right.

**Careful:** Don't use PDL( ) twice in the same line, or even in lines that are next to each other.

In the program above, lines 40 and 48 have PDL( ) in them. They are separated by lines 45 and 47.

(The reason is this: it takes a little time for the paddle to figure out where it is, and if the computer asks again about the paddles too soon, it gets mixed up.)

## THE PUSH BUTTONS

To see if the push button on paddle 0 is being pushed, look at memory location -16287. That is what PEEK(-16287) does. If the number it finds is larger than 127 then the button is being pushed.

The push button on paddle 1 is tested by looking at a nearby memory location, -16286.

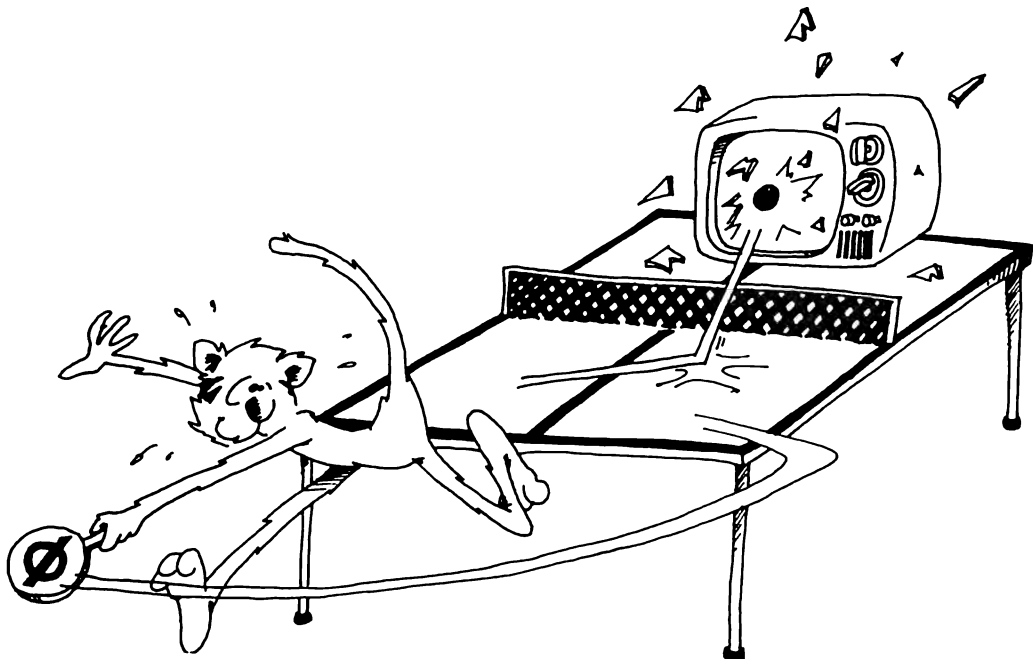
## MOVING A SPOT ON THE SCREEN

```
Run:      10 REM MOVE A SPOT
          15:
          20 HOME:GR
          30 PRINT "USE PADDLE 0"
          50 Y=39
          55 COLOR = 0
          60 PLOT X,Y
          65 X=PDL(0)*39/255
          67 COLOR = 3
          70 PLOT X,Y
          99 GOTO 55
```

Use the RESET key to stop the program. Save to disk using file name "MOVE A SPOT."

Remember the graphics screen is 40 squares high and 40 squares wide. But you number the squares from 0 to 39 each way.

Line 50 says: "put the dot on the last line, at the bottom of the screen."



## ERASE AND PUT

"Erase and put, erase and put, erase and put . . ." Every time you put a dot, you have to erase it again before putting it somewhere else. Otherwise, you will get more and more dots. (To see this happen, change line 55 to COLOR=3, or remove line 60.)

Lines 55 and 60 do the erasing. Line 55 makes the color black, the same as the background color of the screen. Line 60 plots this black dot where the colored dot now is.

Line 67 makes the dot have color 3, which is blue. Then line 70 plots the dot on the screen.

Line 65 asks the paddle where on the bottom line the dot should go. The paddle answers with a number between 0 and 255. But the screen only goes from 0 to 39. So we divide by 255 to get a number between 0 and 1, and then multiply by 39 to get a number between 0 and 39. We call this number X.

## SHOOT A LASER

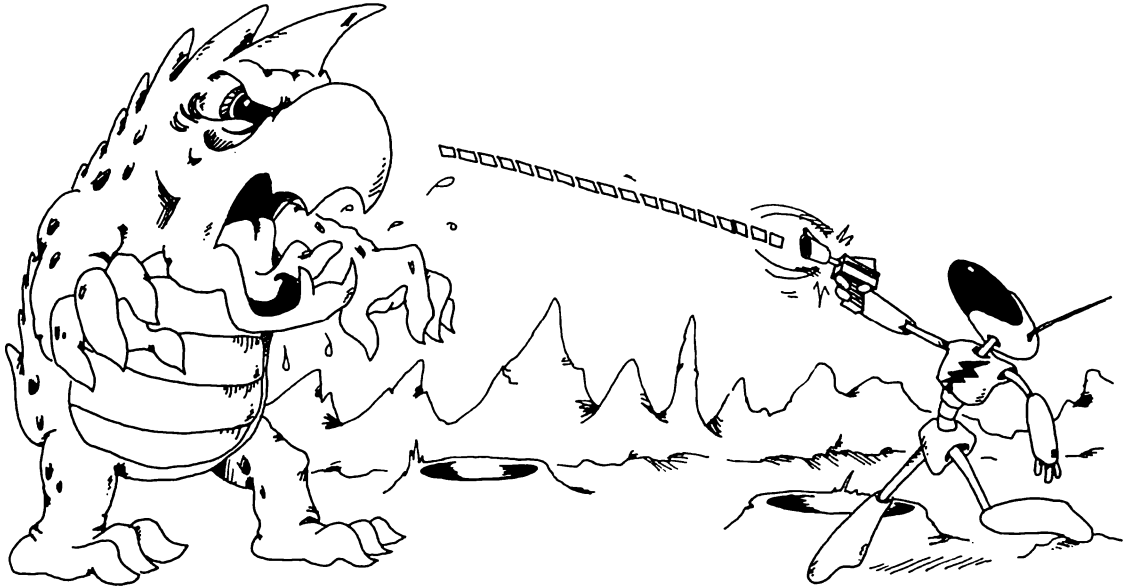
Load program "MOVE A SPOT" and add lines to get:

```
10 REM SHOOT A DOT
20 HOME:GR
30 PRINT "USE PADDLE 0"
50 Y=38
55 COLOR=0
60 PLOT X,Y
65 X=PDL(0)*39/255
67 COLOR = 7
70 PLOT X,Y
75 IF PEEK(-16287)>127 THEN GOSUB 200
99 GOTO 55
200 REM SHOOT
205 COLOR=12
210 FOR I=38 TO 0 STEP -1
220 PLOT X,I
230 NEXT I
240 FOR I=38 TO 0 STEP -1
245 COLOR = 0:PLOT X,I
250 NEXT I
299 RETURN
```

Run. Use the RESET key to end the program. Save to disk in a file named "SHOOTING."

Line 75 asks if the button on the paddle is being pressed. If yes, then the subroutine at line 200 shoots a colored line up and then erases it.

## SHOOTING STARS, THE SCRN( ) FUNCTION



Load the SHOOTING program and add these lines:

```
10 REM *** SHOOTING STARS ***
40 GOSUB 300
215 IF SCRN(X,I)<>0 THEN PRINT CHR$(7);
300 REM INITIALIZE STARS
305 COLOR = 1
310 FOR I=1 TO 10
320 X=RND(8)*40
330 Y=RND(8)*20
350 PLOT X,Y
360 NEXT I
399 RETURN
```

Run the program. Use the RESET key to stop the program. Save it to disk under file name "SHOOTING STARS."

### DRAWING THE STARS

The program calls the subroutine at 300 just once. It draws 10 stars at random near the top of the screen.

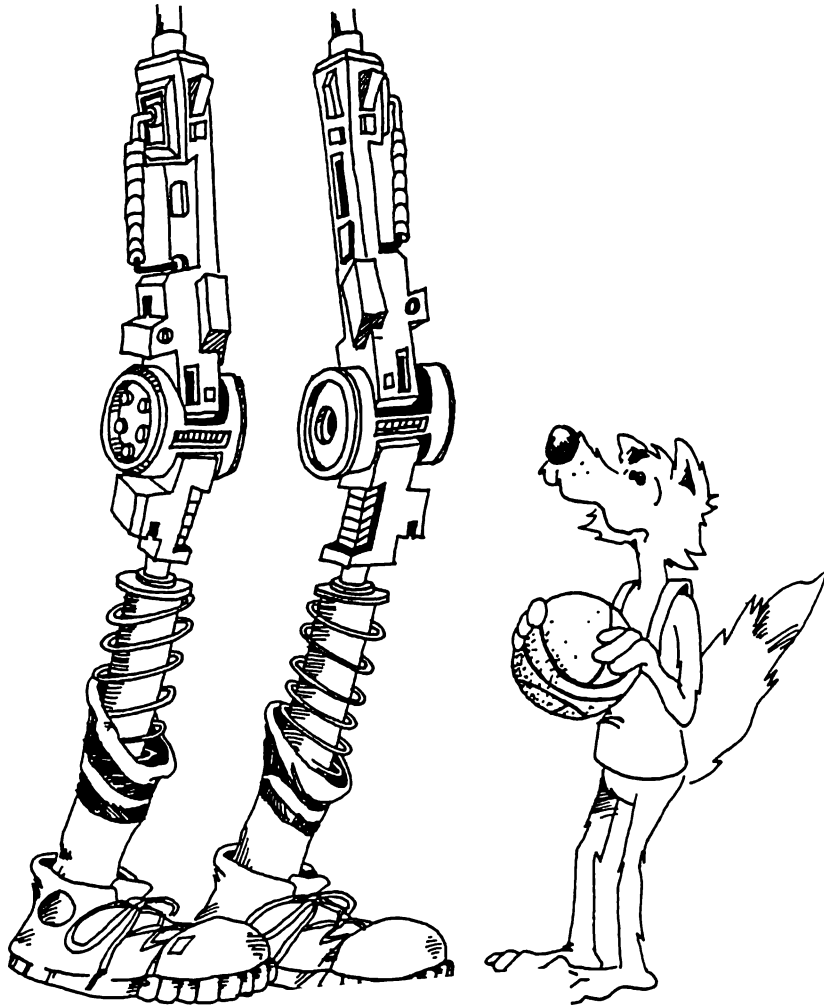
### HAS THE LASER HIT A STAR?

While the program is drawing the laser line, it looks to see if the next square is background (black color). If not, it must be a star, and the beep sound is made to show that you hit a star.

It looks at the square by using the function SCRN(X,Y) in line 215. SCRN(X,Y) asks what color is on the screen at location X,Y. In our case, it is color 0 for the background or color 9 for squares that are stars.

### Assignment 28:

1. The subroutine at line 200 in "SHOOTING" draws the whole laser line before erasing it. Rewrite the subroutine so that each square is erased before the next square is written.
2. In the program "SHOOTING STARS":  
Make the stars of two colors, red and blue, (in the subroutine at line 300) and then in subroutine at 200, let the laser "pop" only the red ones.



## **INSTRUCTOR NOTES 29    ASCII CODE, KEYBOARD, ON . . . GOTO**

This lesson treats the ASCII code for characters, and the functions `ASC( )` and `CHR$( )` that change characters to numbers and vice versa.

The ASCII code is primarily intended to standardize signals between hardware pieces such as computers with printers, terminals, other computers, etc. But within programs the ASCII numbers also are useful. The letters are numbered in increasing order and so the ASCII numbers are useful in alphabetizing routines. The numerical digits are also in order, and the punctuation marks also have ASCII numbers.

Some of the signals sent between hardware, such as “bell,” “line feed,” and “carriage return” also have ASCII numbers which can be used inside of `PRINT` statements for control of the TV screen and the Apple’s loud speaker.

### **QUESTIONS:**

1. Does `ASC(S$)` return a string or a number for its value?
2. Does `ASC(S$)` have a string or a number for its argument?
3. Same two questions for `CHR$(N)`.
4. Which letter has the larger ASCII code number, B or W?
5. Do you know the ASCII code for the character “1”? Is it the number 1?
6. What will the computer do if you run this line:

```
10 PRINT CHR$(13); CHR$(7)
```

(If you don’t know, try it.)



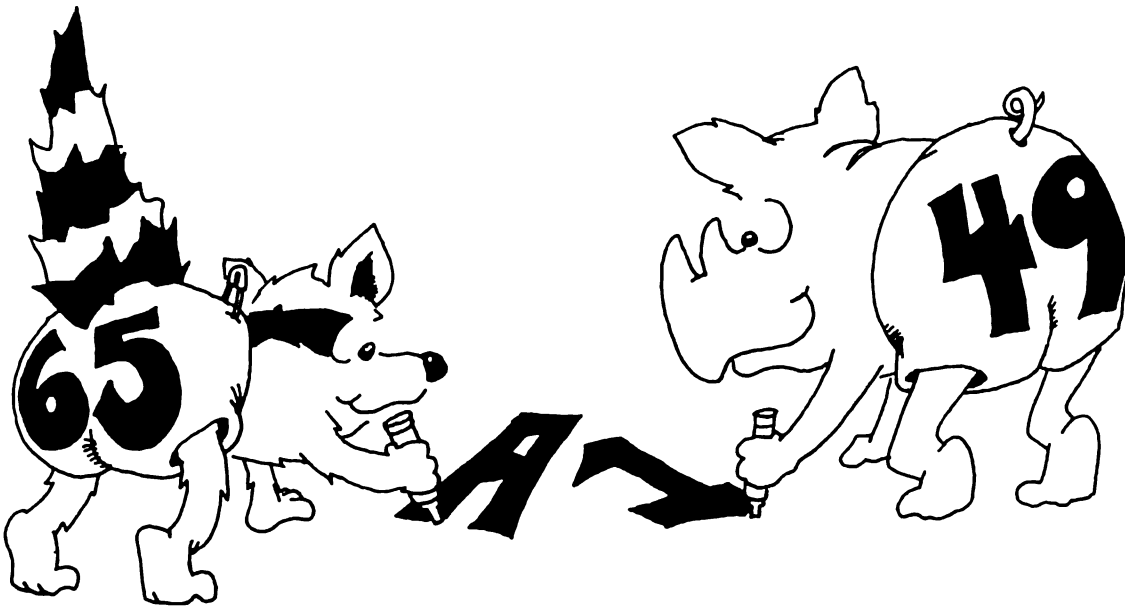
## LESSON 29 ASCII CODE, KEYBOARD, ON . . . GOTO NUMBERING THE LETTERS IN THE ALPHABET

"That is easy," you say. "A is 1, B is 2, C is 3 . . ."

Well, for some strange reason, it goes like this: A is 65, B is 66, C is 67 . . . .

These numbers are called the ASCII code of the characters. ASCII is pronounced "ask-key."

The punctuation marks and number digits have ASCII code numbers too.



### ASC( ) CHANGES CHARACTERS INTO NUMBERS

Use the ASC( ) function to change characters into ASCII numbers.

```
Run:      10 REM *** WHAT NUMBER IS THIS KEY? ***
          20 PRINT
          25 PRINT "PRESS KEYS TO SEE ASCII NUMBER"
          30 GET C$
          40 PRINT C$;TAB(5);ASC(C$)
          50 GOTO 30
```

Try out some letters, digits, and punctuation. Try also the RETURN key and other keys.

Press RESET to end the program. Then SAVE it to disk.



## ALPHABETICAL LIST

What good are the ASCII numbers? Well, they can help in making alphabetical lists.

```
Run:  10 REM ALPHABETIZE
      20 PRINT
      30 INPUT"GIVE ME A LETTER: ";A$
      35 PRINT
      40 INPUT"GIVE ME ANOTHER: ";B$
      45 A=ASC(A$):B=ASC(B$)
      47 REM PUT IN ALPHABETICAL ORDER BY
      48 REM SEEING WHICH HAS THE LOWER ASCII NUMBER.
      50 IF A>B THEN X=A:A=B:B=X:REM SWAP THEM
      55 PRINT
      60 PRINT"HERE THEY ARE IN ALPHABETICAL ORDER"
      65 PRINT:PRINT CHR$(A);TAB(5);CHR$(B)
```

Save it to disk.

Look at these two functions: ASC( ) and CHR\$( ).

ASC( ) gives you the ASCII number for the FIRST character in the string.

CHR\$( ) does the reverse. It gives you the character belonging to each ASCII number.

## THE ASCII NUMBERS FOR CHARACTERS

The ASCII numbers for letters of the alphabet start at number 65 for A and go to 90 for Z.

The digits start with 48 for zero and go to 57 for the digit "9." Punctuation starts at 32 for "space" and goes to 47. It starts again at 58 and goes to 64. It starts once more at 91 and goes to 95.

The ASCII code for RETURN is 13.

## CHANGING NUMBERS INTO CHARACTERS

Use CHR\$( ) to change ASCII code numbers into a string holding one character.

```
Run:  10 REM /// DISPLAY ASCII ///
```

```
      11 :
```

```
      20 HOME
```

```
      25 PRINT "NOTE: FIRST 32 CHARACTERS ARE INVISIBLE"
```

```
      30 FOR I=0 TO 255
```

```
      40 PRINT I, CHR$(I)
```

```
      50 FOR T=1 TO 500:NEXT T
```

```
      60 NEXT I
```

Save the program to disk.

At 96 the whole shebang starts over, so you really do not need to go above 95 in line 30.

Nothing gets printed for numbers 0 through 31. But number 7 makes the Apple peep and numbers 10 and 13 make it skip a line.

The ASCII number 7 stands for "bell." When printed, it makes the Apple peep.

The ASCII number 10 stands for "line feed." When sent to the screen or to a printer, it moves the cursor to the next line down.

The ASCII number 13 stands for "carriage return." The name comes from the carriage return on a typewriter. It moves the cursor to the beginning of the line.

(Actually, the carriage return lever on a typewriter does a "line feed" followed by a "carriage return.")

## GAMES AND THE KEYBOARD

The GET command makes the computer wait for you to press a key. The program stops running until you press a key.

There is another way to get a keystroke from the keyboard that does not make the computer wait. It is used in action games. NOTE: The Snake program has bugs; see page 166.

```
2   GOTO 1000: REM    SNAKE
100  REM MAIN LOOP
101  :
102  CL = CL + 11
103  IF CL > 12 THEN CL = 1
104  COLOR = CL
105  DR = PEEK (AR)
107  S = PEEK (ST)
110  IF DR = LL THEN D = D - 1: IF D = 0 THEN D = 4
111  IF DR = R THEN D = D + 1: IF D = 5 THEN D = 1
113  FOR T = 1 TO 50: NEXT T
115  ON D GOTO 120,122,124,126
120  Y = Y - 1: GOTO 130
122  X = X - 1: GOTO 130
124  Y + 1: GOTO 130
126  X = X + 1
130  PLOT X,Y
140  A = B: B = C: C = E: E = F: F = G: G = X
142  L = M: M = N: N = O: O = P: P = Q: Q = Y
145  COLOR = 0: PLOT A,L
199  GOTO 100
999  END
1000 :
1001  REM          *****SNAKE*****
1002 :
1010  REM          BY EDWARD H. CARLSON
1011 :
2000  REM  BORDER
2010  GR : COLOR = 7
2020  HLIN 0,39 AT 0
2022  VLIN 0,39 AT 39
2024  HLIN 0,39 AT 39
2046  VLIN 0,39 AT 0
2100  LL = 149
2102  R = 136
2105  AR = 49152
2107  STROBE = 49168
2110  X = 20: Y = 20
2115  A = X: B = X: C = X: E = X: F = X: G = X
2116  L = Y: M = Y: N = Y: O = Y: P = Y: Q = Y
3000  REM  INSTRUCTIONS
3010  INPUT "DO YOU WANT INSTRUCTIONS? ";Y$
```

```

3015 IF Y$ < > "Y" THEN 3999
3020 PRINT "TURN LEFT, LEFT ARROW KEY"
3022 PRINT "TURN RIGHT, RIGHT ARROW KEY"
3999 REM
9990 COLOR= 1
9999 GOTO 100

```

## THE KEYBOARD'S BOX

Every time a key is pressed, the computer puts the character in a box with the name "49152" on the front. This is where INPUT goes to find characters. This is where GET goes to find characters. We can look in the box too.

To look in the box, use the PEEK command. The SNAKE program used lines:

```

105 DR=PEEK(AR)
107 S= PEEK(ST)

```

where AR=49152 and ST=49168. Using variable names inside PEEK instead of constant numbers makes the program run faster.

Line 105 looks in the keyboard's box to see if a key has been pressed lately. If so, the ASCII number of the key is taken from the keyboard box and put in box DR.

Line 110 asks if the key was the left arrow (ASCII number 149, see line 2100).

What does line 111 do? \_\_\_\_\_

Line 107 tells the computer to empty the keyboard's box so it is ready for another character from the keyboard. It does this using:

```

107 S=PEEK(ST)

```

Where does 49168 get put in box ST?

## THE ON ... GOTO COMMAND

```

115 ON D GOTO 120,122,124,126

```

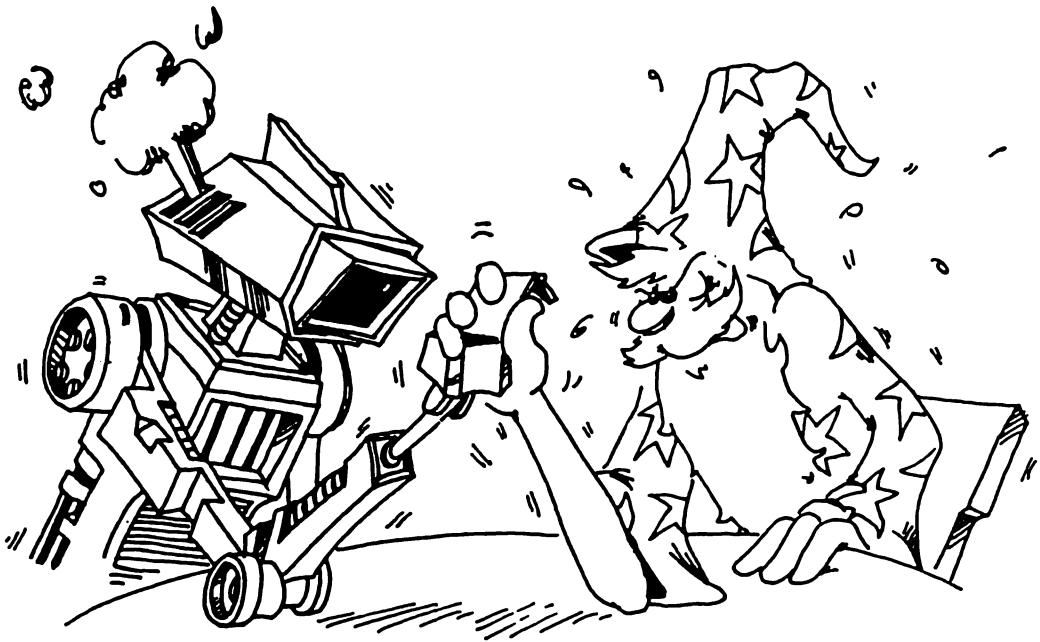
This means that:

if D is	1	GOTO	120
	2		122
	3		124
	4		126
if D is something else		GOTO	the next line

After the GOTO, you can put one, two, or as many numbers as you want. Each number is the same as the number of a line somewhere in the program.

### Assignment 29:

1. Write a program which asks for a word. Then it rearranges all the letters in alphabetical order.
2. Write a program that speaks "double dutch." It asks for a sentence, then removes all the vowels and prints it out.
3. Write a program that uses GET to get a letter A to C to use in a menu. Change the letter to a number 1 to 3. Then use the ON . . . GOTO command to pick which menu item to do.



## **INSTRUCTOR NOTES 30   ARRAYS AND THE DIM COMMAND**

This lesson introduces arrays. The DIM( ) statement is described.

Arrays with one index are described first. The array itself is compared to a family, and the individual elements of the array to family members, with the index value being the “first name” of the member.

Two dimensional arrays are compared to the rectangular array of cells on the TV screen.

Higher dimensional arrays are just mentioned, with no examples given.

### **QUESTIONS:**

1. What does the DIM AD\$(5) command do?
2. Where do you put the DIM command in the program?
3. What two kinds of array families are there?
4. What is the “index” or “subscript” of an array?
5. What does the command DIM SR(5,9) do?

## LESSON 30 ARRAYS AND THE DIM COMMAND

### MEET THE ARRAY FAMILY

```
22 F$(0) = "DAD"  
24 F$(1) = "MOM"  
26 F$(2) = "BRIAN"
```

Each member of the family is a variable. The F\$ family are string variables.

Here is a family of numerical variables:

```
35 N(0) = 43  
37 N(1) = 13  
39 N(2) = 0  
41 N(3) = 0
```

The family has a “last name” like A( ) or B\$( ). Each member has a number in ( ) for a “first name.” The array always starts with the first name “0”.

Instead of “family” we should say “array.”

Instead of “first name” we should say “index number” or “subscript.”

### THE DIM() COMMAND RESERVES BOXES

When the array family goes to a movie, they always reserve seats first. They use a DIM command to do this.

The DIM . . . command tells the computer to reserve a row of boxes for the array. DIM stands for “dimension” which means “size.” For example, the statement

```
18 DIM A(3)
```

saves four memory boxes, one each for the variables A(0), A(1), A(2), and A(3). These boxes are for numbers and contain the number “0” to start with. Another example:

```
30 DIM A(3),B$(4)
```

This time, DIM reserves 4 boxes for the A( ) array and 5 for the string array B\$( ). The boxes named B\$(0) through B\$(4) are for strings and are empty to start with.

**Rule:** Put the DIM( ) statement early in the program, before the array is used in any other statement.





## MAKING A LIST

Enter:

```

10 REM +++ IN A ROW +++
20 HOME:PRINT
30 DIM A$(5)
35 PRINT"ENTER A WORD"
40 FOR N=0 TO 5
45 IF N>0 THEN PRINT"ANOTHER"
50 INPUT A$(N)
55 PRINT
60 NEXT N
70 PRINT
100 REM PUT IN A ROW
105 PRINT"HERE THEY ARE IN A ROW"
106 PRINT
110 FOR I=0 TO 5
120 PRINT A$(I);" ";
130 NEXT I

```

Run and save to disk.

You can use a member of the array by itself, look at this line:

```
40 B$(2)="YELLOW SUBMARINE"
```

Or the array can be used in a loop where the index keeps changing. Lines 50 and 120 in the program "IN A ROW" do this.

## MAKING TWO LISTS

Enter:

```
10 REM PHONE LIST
20 HOME:PRINT
30 DIM NAME$(20), NUMBER$(20)
35 I=0
40 PRINT "ENTER NAMES AND NUMBERS "
50 PRINT: INPUT "NAME? ";NA$(I)
60 INPUT "NUMBER? ";NU$(I)
70 I=I+1:GOTO 50
```

Run. Press RESET key to stop program. Save to disk.

## ONE DIMENSION, TWO DIMENSION, . . .

The arrays that have one index are called one dimensional arrays. But arrays can have 2 or more indices. Two dimensional arrays have their "family members" put in a rectangle like the days in a month on a calander.

```
10 REM +++ TWO-DIM ARRAY +++
15 :
18 HOME
20 DIM T(5,6)
30 FOR X=0 TO 5
40 FOR Y=0 TO 6
50 T(X,Y)=X+Y
60 NEXT Y,X
65 :
70 REM ***** PRINT OUT THE ARRAY
72 :
80 FOR J=0 TO 6
82 VTAB(5+3*J)
85 FOR I=0 TO 5
87 HTAB(3+4*I):PRINT T(I,J);
90 NEXT I,J
```

## Assignment 30:

1. Finish the PHONE LIST program so that it prints out the list of names with the telephone numbers beside them.
2. Use a two dimensional array to make a "weekly calendar" program. It could use an array made by DIM AR\$(7,24) so that each day of the week could have an entry for each hour.

## INSTRUCTOR NOTES 31 LOGIC: AND, OR, NOT

This lesson treats the AND, OR, and NOT relations and the numerical values for TRUE and FALSE. These are important for some types of IF statements.

The TEENAGER program in lesson 12 used a nested IF to print out "You are a teenager." A more concise logic uses the OR relation.

There are several abstract ideas in this lesson that are difficult to grasp. The fact that TRUE and FALSE have numerical values of 1 and 0 is bad enough. But in addition, the computer often treats any number that is not zero as being TRUE.

### QUESTIONS:

1. For each IF statement, tell if it will print anything:

```
10 IF 3=3 THEN PRINT "HI "  
10 IF NOT(3=3) THEN PRINT "HI "  
10 IF 3=3 OR 0=2 THEN PRINT "HI "  
10 IF 3=3 AND 0=2 THEN PRINT "HI "  
10 IF "A"="B" THEN PRINT "HI "  
10 IF NOT("A"="B") THEN PRINT "HI "
```

2. What number will each of these lines print?

```
10 A=1:PRINT A, NOT A  
10 A=0:PRINT A, NOT A  
10 A=1:B=1:PRINT A AND B  
10 A=0:B=1:PRINT A AND B  
10 A=0:B=0:PRINT A AND B  
10 A=0:B=1:PRINT A OR B  
10 A=0:B=0:PRINT A OR B  
10 PRINT NOT 23  
10 PRINT NOT 0  
10 PRINT 3 AND 7  
10 PRINT 3 AND 0
```

## LESSON 31 LOGIC: AND, OR, NOT

### ANOTHER TEENAGER PROGRAM

```
Enter:  10 REM <<< AND , OR , NOT >>>
        20 HOME:PRINT
        30 INPUT"YOUR FIRST NAME ";N$
        35 PRINT
        40 INPUT"YOUR AGE ";A
        45 PRINT
        50 IF (A>12) AND (A<20) THEN PRINT N$" IS A TEENAGER."
        55 NFLAG = (A<13) OR (A>19)
        60 IF NFLAG THEN PRINT N$ " IS NOT A TEENAGER."
        65 PRINT
        70 IF (NOT NFLAG) AND (A=16) THEN PRINT "AND "N$" IS
           SWEET SIXTEEN."
```

Run and save to disk.

### WHAT DOES "AND" MEAN?

Two things are true about teenagers: they are over 12 years old and they are less than 20 years old. Look at line 50.

IF (you are over 12) AND (you are less than 20) THEN (you are a teenager).

### WHAT DOES "OR" MEAN?

In line 55 the OR is used. Two things are said: "age is under 13" and "age is over 19."

Only one of them needs to be true for you to be "not a teenager."

IF (you are under 13) OR (you are over 20) THEN (you are not a teenager).

### TRUE AND FALSE ARE NUMBERS

How does the computer do it? It says true and false are numbers.

Rule:                      TRUE is the number      1

                              FALSE is the number      0

(It is easy to remember that 0 is FALSE because zero is the grade you get if your homework is false.)

To see these numbers, enter this in the edit mode:

```
PRINT 3=7
```

The computer checks to see if 3 really does equal 7. It doesn't so it prints a "0" meaning FALSE.

And this: 

```
PRINT 3=3
```

The computer checks to see if  $3=3$ . It is, so the computer prints "1" meaning "TRUE."

### PUTTING TRUE AND FALSE IN BOXES

The numbers for TRUE and FALSE are treated just like other numbers and can be stored in boxes with numerical variable names on the front. Run this:

```
10 N= ( 3=22 )  
20 PRINT N
```

The number 0 is stored in the box N because  $3=22$  is FALSE.

And this: 

```
10 N= "B"="B"  
20 PRINT N
```

The number 1 is stored in the box N because the two letters in the quotes are the same so the statement "B"="B" is TRUE.



## THE IF COMMAND TELLS LITTLE WHITE LIES

The IF command looks like this:

```
10 IF (something A) THEN (command C)
```

Try these in the edit mode:

```
IF 0 THEN PRINT "TRUE"
```

```
IF 1 THEN PRINT "TRUE"
```

Now try this:

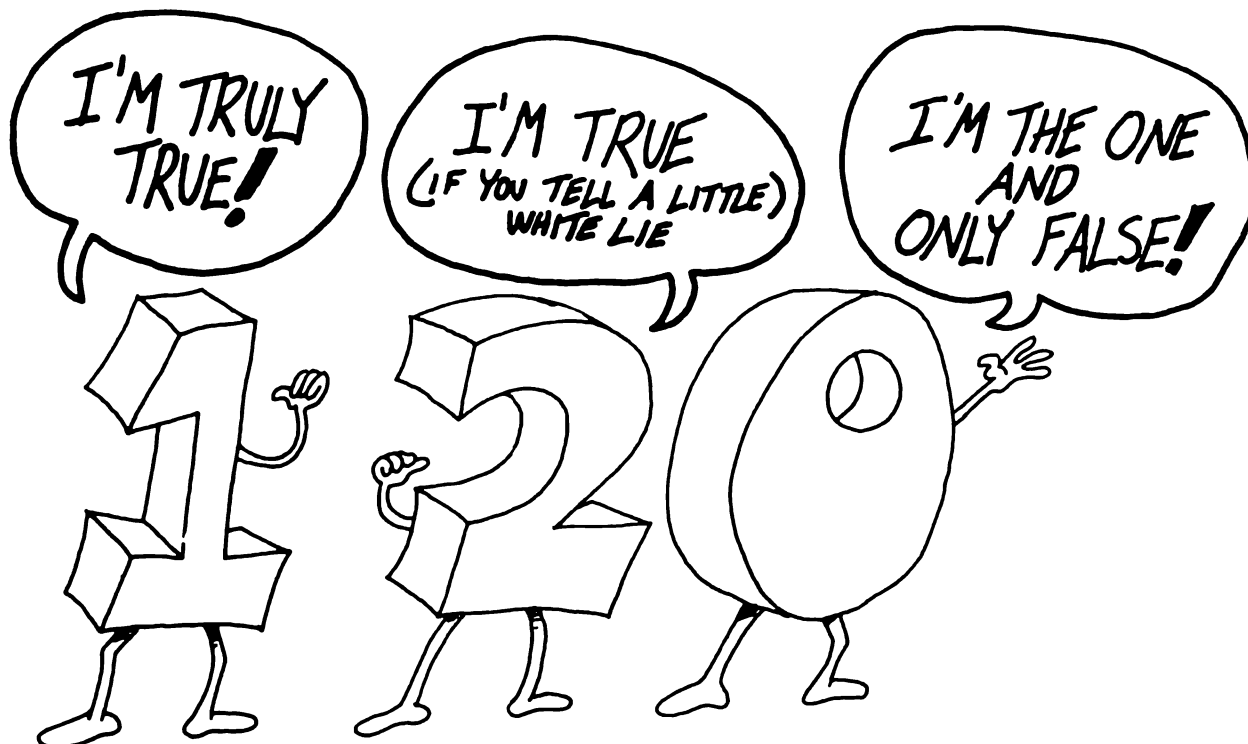
```
IF 22 THEN PRINT "TRUE"
```

**Rule:** In an IF, the computer looks at "something A."

If it is zero, the computer says "something A is FALSE," and skips what is after THEN.

If it is not zero, the computer says "something A is TRUE" and obeys the commands after THEN.

The IF command tells little white lies. TRUE is supposed to be the number "1," but the IF stretches the truth to say "TRUE is anything that is not FALSE." That is, any number that is not zero is TRUE.

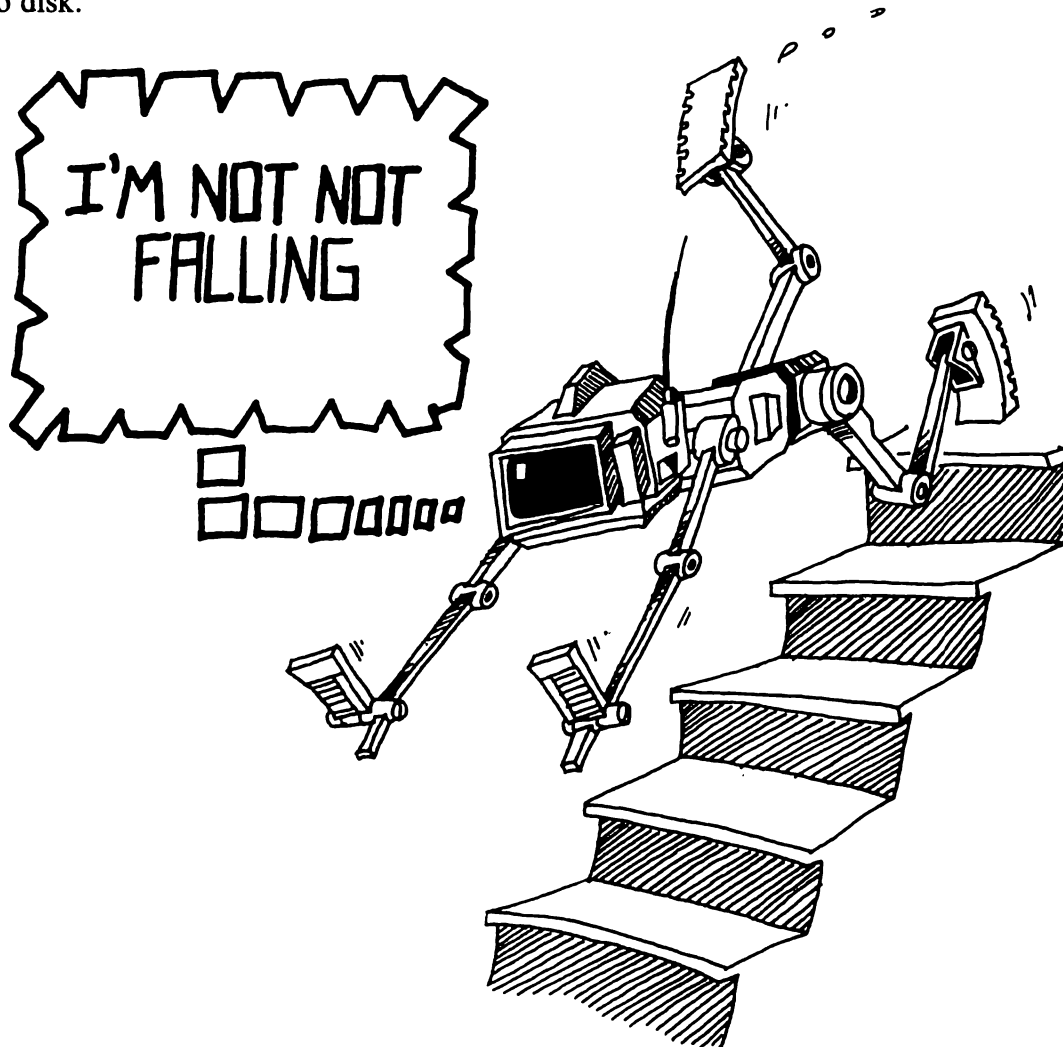


## WHAT DOES "NOT" MEAN?

NOT changes FALSE to TRUE and TRUE to FALSE. Try this:

```
10 REM ??? DOUBLE NEGATIVE ???  
20 N=3  
30 PRINT "N ";      TAB(20); N  
40 PRINT "NOT N";    TAB(20); NOT N  
50 PRINT "NOT NOT N ";TAB(20); NOT (NOT N)  
60 REM The computer knows that "I don't have no..."  
61 REM means "I do have ...."
```

Save to disk.



The NOT also tells little white lies:

N was 3, which is called TRUE, (a little white lie.)

Then NOT 3 is FALSE, or the number 0.

Finally, NOT (NOT 3) is the same as NOT (0) or NOT (FALSE) or TRUE or the number 1.

## THE LOGICAL SIGNS

You can use these 6 symbols in the "something A" phrase:

=	equal
< >	not equal
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal

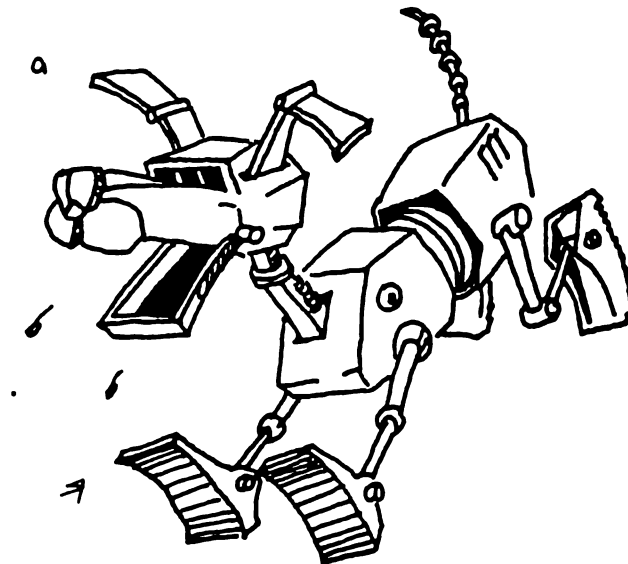
You have to press two keys to make the < > sign and the <= and >= signs.

The last two are new, so look at this example to see the difference between < and <= :

2<=3	is	TRUE	2<3	is	TRUE
3<=3	is	TRUE	3<3	is	FALSE
4<=3	is	FALSE	4<3	is	FALSE

These two "something A" phrases mean the same:

$$2<=Q \quad (2<Q) \quad \text{OR} \quad (2=Q)$$





### Assignment 31:

1. Tell what will be found in the box N if:

```
N=4=4
N="G"<>"S"
N=5>7
N=3>2 AND 3<2
N=4=3 OR 4=4
N=NOT 0
N=5>=4
```

2. Tell if the word "JELLYBEAN" will be printed:

IF 0	THEN PRINT "JELLYBEAN"
IF 1	THEN PRINT "JELLYBEAN"
IF 9	THEN PRINT "JELLYBEAN"
IF 3<>0	THEN PRINT "JELLYBEAN"
IF 2 AND 4	THEN PRINT "JELLYBEAN"
IF 0 OR 1	THEN PRINT "JELLYBEAN"
IF NOT 3	THEN PRINT "JELLYBEAN"
IF "A"="Z"	THEN PRINT "JELLYBEAN"
IF NOT(3) AND 2	THEN PRINT "JELLYBEAN"
IF NOT(0) OR 0	THEN PRINT "JELLYBEAN"
IF 4<=5	THEN PRINT "JELLYBEAN"

3. Write a program to detect a double negative in a sentence. Look for negative words like not, no, don't, won't, can't, nothing and count them. If there are 2 such words there is a double negative. Test the program on the sentence "COMPUTERS AIN'T GOT NO BRAINS."

## **INSTRUCTOR NOTES 32 USER FRIENDLY PROGRAMS**

This lesson concerns clear programs which interact with the user in a “friendly” way.

The “spaghetti” program should be discouraged. A format for writing programs is presented in this lesson. While methods of imposing order on the task are largely a matter of taste, the methods used in this lesson can serve to introduce the ideas.

“User friendly” means that the screen displays are easy to read, keyboard input is “RETURN key free” as much as possible, and errors are “trapped.” Ask if entries are ok. If not, give an opportunity to fix things.

Instructions and “HELP” should be available. Prompts need to be given. Beginners need complete prompts, but experienced users would rather have curt prompts.

It is hard to teach the writing of “user friendly” programs. Success depends mostly on the attitude of the programmer. The best advice is to “turn up your annoyance detectors to high” as you write and debug the program.

Most young students will not progress very far toward fully “friendly” programming. To be acquainted with the desirability of “friendly” programming and to use some simple techniques toward accomplishing it are satisfactory achievements.

### **QUESTIONS:**

1. Should your program give instructions whether the user wants them or not?
2. What is a “prompt”? Give two examples.
3. What is “scrolling”? How can you write to the screen without scrolling?
4. If you want the user to enter a single letter from the keyboard, what command is best? (Avoid using the RETURN key.)
5. What is an “error trap”? How would you trap errors if you asked your user to enter a number from 1 to 5?
6. In what part of the program are most of the GOSUB commands found?
7. Why put the “STARTING STUFF” section of the program at the end of the program (at high line numbers)?

## LESSON 32 USER FRIENDLY PROGRAMS

There are two kinds of users:

1. Most want to run the program. They need:

- instructions
- prompts
- clear writing on the screen
- no clutter on the screen
- erasing old stuff from the screen
- not too much key pressing
- protection from their own stupid errors

2. Some want to change the program. They need:

- a program made in parts
- each part with a title in a REM
- explanations in the program

(Don't forget you are a user of your own programs, too! Be kind to yourself!)

### PROGRAMS HAVE THREE PARTS

“STARTING STUFF”: at the beginning of the program run

- give instructions to the user
- draw a screen display
- set variables to their starting values
- ask the user for starting information

MAIN LOOP:

- controls the order in which tasks are done
- calls subroutines to do the tasks

SUBROUTINES:

- do parts of the program



## PROGRAM OUTLINE

```

1 GOTO 1000:REM *** Program name ***
...
100 REM MAIN LOOP
...
...   calls subroutines
...
199 END
1000:
1001 REM *** Program name ***
1002:
...   REM's that give a description of the
...   program, variable names, etc.
...
1999:
2000 REM STARTING STUFF
...
...   ask for starting information
...   set variable values
...   give instructions
...
2999 GOTO 100

```

### PUT THE MAIN LOOP AT THE BEGINNING OF THE PROGRAM

Put the MAIN LOOP near the front because it will run faster there.

### PUT STARTING STUFF AT THE END OF THE PROGRAM

Put the STARTING STUFF near the back because it may be the biggest part of the program, and you may keep adding to it as you write, to make the program more “user friendly.” It does not need to run fast.

## PUT SUBROUTINES IN THREE PLACES

between line 2 and line 99 for subroutines that must run fast  
after line 2999 for starting stuff subroutines  
between line 200 and line 99 for the rest of the subroutines

## INFORMATION PLEASE

```
280 PRINT "DO YOU WANT INSTRUCTIONS <Y/N>"
```

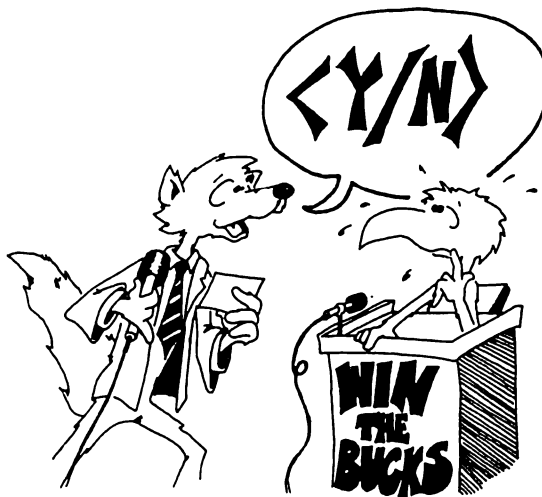
This lets a beginner see instructions, and lets others say "no."

## TIE A STRING AROUND THE USER'S FINGER

Use a "prompt" to remind users what choices they have.

Example: <Y/N> where the choice is Y for "yes" or N for "no"

Beginners need long prompts. Other users like short prompts.



## DON'T GIVE THE USER A HEADACHE

SCROLLING gives headaches!

BASIC usually scrolls. It writes new lines at the bottom of the screen and pushes old lines up.

It is like the scrolls the Romans used for writing. They unwound from the bottom and wound up at the top.

Avoid scrolling. Use VLIN and HLIN to print just where you want. Erase by printing a string of blanks to the same spot.

Use delay loops so the writing stays on the screen while the user reads it.



### **OUCH! MY FINGERS HURT**

Use the GET command to enter single letters. This saves having to press RETURN.

```
380 PRINT "DO YOU NEED INSTRUCTIONS? <Y/N> "  
382 GET R$: IF R$="Y" THEN GOSUB 3400
```

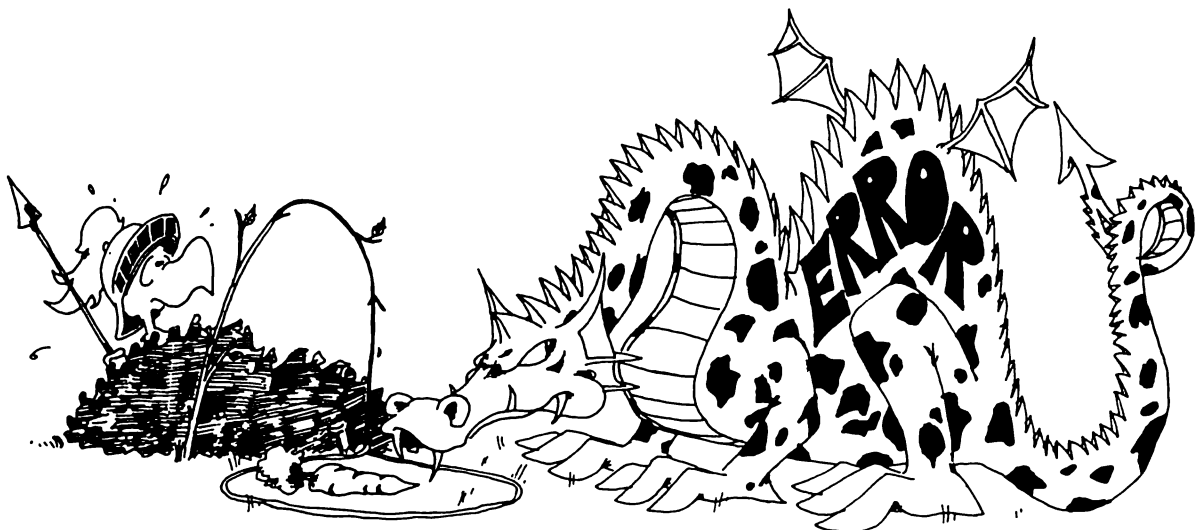
### **SET TRAPS FOR ERRORS**

Example: Add this line to the above lines:

```
384 IF R$<>"N" THEN GOTO 380
```

Line 380 asked for only two choices, Y or N. If the user presses some other key, line 384 sends him back to line 380.

Traps make your program "bomb proof" so that users will be unable to goof it up!



## Assignment 32:

1. Look at the COLOR EATER program. Add REM's to explain the lines in the program. Fix up the information that is printed. For example, the "I HAVE NO FOOD" message ruins the messages printed in lines 2202 and 2210.
2. Write a secret cipher program. The user chooses a password and it is used to make a cipher alphabet like this:

if the password is APPLE

remove the repeated letters, get APLE

put it at the front of the alphabet and the rest of the letters after it in normal order

APLEBCDFGHIJKMNOQRSTUVWXYZ

The user chooses to code or decode from a menu.

```
1  GOTO 1000: REM *** COLOR EATER ***
2  :
100 NC = 0
101 FOR I = X - 1 TO X + 1
102 FOR J = Y - 1 TO Y + 1
104 IF I < 0 THEN I = 0
105 IF I > 39 THEN GOTO 120
106 IF J < 0 THEN J = 0
107 IF J > 39 THEN GOTO 120
109 CC = SCRN( I,J)
110 IF CC = C THEN X = I:Y = J: COLOR= 0: PLOT
X,Y: GOTO 100
112 IF CC < > 0 THEN NC = 1
115 NEXT J,I
120 C = C + 1: IF C > 15 THEN C = 1
125 IF NC = 0 THEN GOSUB 300
130 VTAB (22): HTAB (1): PRINT X;" " TAB ( 5)Y"
";
135 VTAB (24): HTAB (1): PRINT C" ";
199 GOTO 100
300 REM COLOR EATER HAS NO FOOD
310 X = X + 1: IF X > 39 THEN X = 1
320 PRINT "I HAVE NO FOOD!";
330 PRINT CHR$ (7)
335 FOR T = 1 TO 500: NEXT T
340 HOME
399 RETURN
1000 REM
2000 REM STARTING STUFF
2010 HOME : GR
2020 FOR I = 0 TO 39
2030 FOR J = 0 TO 39
2040 C = INT ( RND (8) * 15) + 1
2045 COLOR= C
2050 PLOT I,J
2060 NEXT J,I
2100 X = 20:Y = 20
2200 VTAB 22: HTAB 8: PRINT "X,Y LOCATION";
2210 VTAB 24: HTAB 8: PRINT "EATING COLOR";
2999 GOTO 100
```

## **INSTRUCTOR NOTES 33    DEBUGGING, STOP, CTRL-C, CONT**

It is difficult to drill systematically on debugging, unless you are NASA with NASA's budget and time scale.

We present a series of small techniques and a description of how to put them together in a debugging scheme. Only practice will serve to make debugging a chore that the student approaches with some confidence.

### **QUESTIONS:**

1. What two ways can you make the computer print

**BREAK IN 55**

while the program is running?

2. How are the STOP and the END commands different?
3. How are the STOP and CTRL-C commands different?
4. What does the CONT command do?
5. Why would you put STOP commands in your program?
6. How do delay loops help you debug a program?
7. How do extra PRINT commands help you debug a program?
8. Why do you take the STOP and extra PRINT commands out of the program after you have fixed the errors?
9. Can you pick in what line the CTRL-C command will stop the program? Can you pick using the STOP command?



## LESSON 33 DEBUGGING, STOP, CTRL-C, and CONT

### THE STOP COMMAND

Enter and run:

```
10 REM SECRET STOP
20 HOME
25 R=INT(RND(8)*200)
30 FOR I=0 to 200
40 IF I=INT (RND(R)*200) THEN STOP
50 NEXT I
```

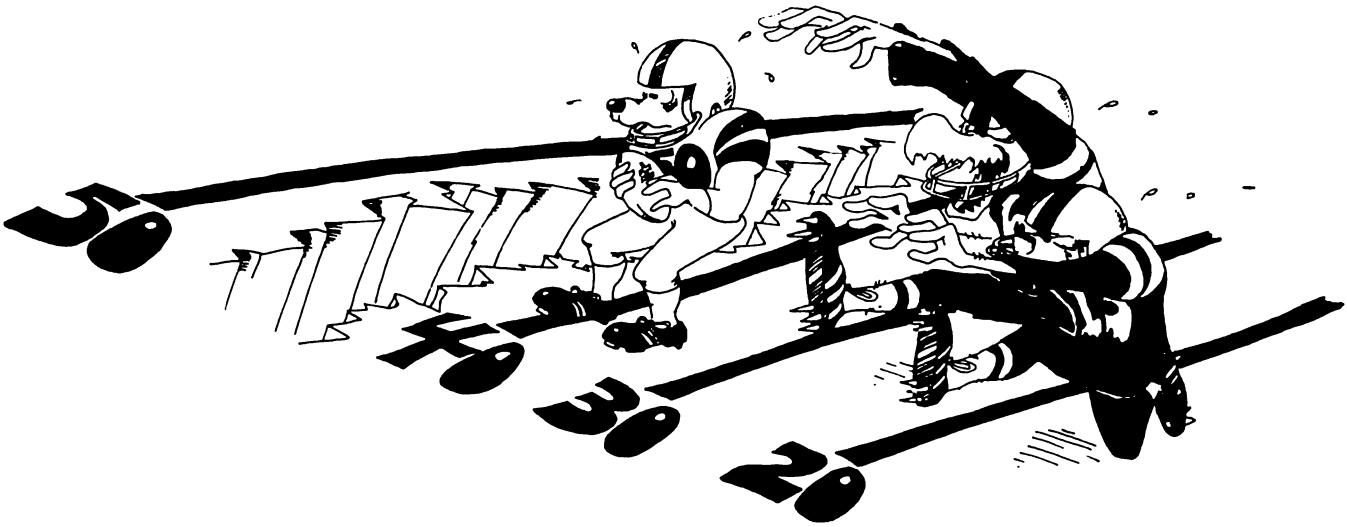
The program will stop, and the computer will beep and print a message:

```
BREAK IN 40
```

What do you suppose the secret value of I was?

Enter:                    PRINT I                    (No line number)

and find out.



### HOW TO START IT AGAIN

Enter the command CONT. Try it!

### “STOP” IS LIKE “END”

STOP makes the computer stop and enter the edit mode.

It is like END except it beeps and prints the number of the line that the STOP is in.

You can have as many STOP commands in your program as you like.

STOP is used for debugging your program.

## **ANOTHER WAY TO STOP RUNNING THE PROGRAM**

You can stop running the program with "CTRL-C." This means you hold down the key that says CTRL on it, and then press the "C" key.

Try it:

```
10 REM GO FOREVER
15 PRINT:SPEED = 100
20 PRINT "MUD TURTLES OF THE WORLD"
30 PRINT "UNITE!":PRINT
35 SPEED = 255
40 FOR T=1 TO 1000:NEXT T
99 GOTO 10
```

The command CTRL-C stops the program at whatever spot it is. It prints:

```
BREAK IN LINE XX      peeps and enters the edit mode
```

(where XX is the line number where it stops.)

The command CONT starts the program again at the same spot.

Try this:                      Try to make the above program stop in line 20. Then make it stop in line 30, then line 40.

## **WHAT DO YOU DO AFTER YOU STOP?**

You put STOP in whatever part of your program is not working right. Then you run the program. After it stops, you look to see what happened.

(Or you use CTRL-C to stop the program, but it may not stop in the spot where the trouble is.)

Put on your thinking cap. Ask yourself questions about what happened as the program ran.

You are in the edit mode. You can:

    List parts of the program and study them.

    Use the PRINT command to look at variables. Do they have the values you expected?

If you find the trouble, you may add a line, change a line, or delete a line.

## STARTING THE PROGRAM AGAIN

There are four ways to start a program. They are:

CONT	if you have not changed the program
GOTO XX	where XX is a line number
RUN XX	where XX is a line number
RUN	your old friend

You may use the CONT command if you have not:

- added a line
- deleted a line
- or changed a line by editing it

Or you may start running the program at a different spot by entering (without line number) the command:

GOTO XX

where XX is the line number where you want to restart .

If you have changed the program, your only choice is to start at the beginning or at some other line number XX with RUN.

What is the difference between these four ways?

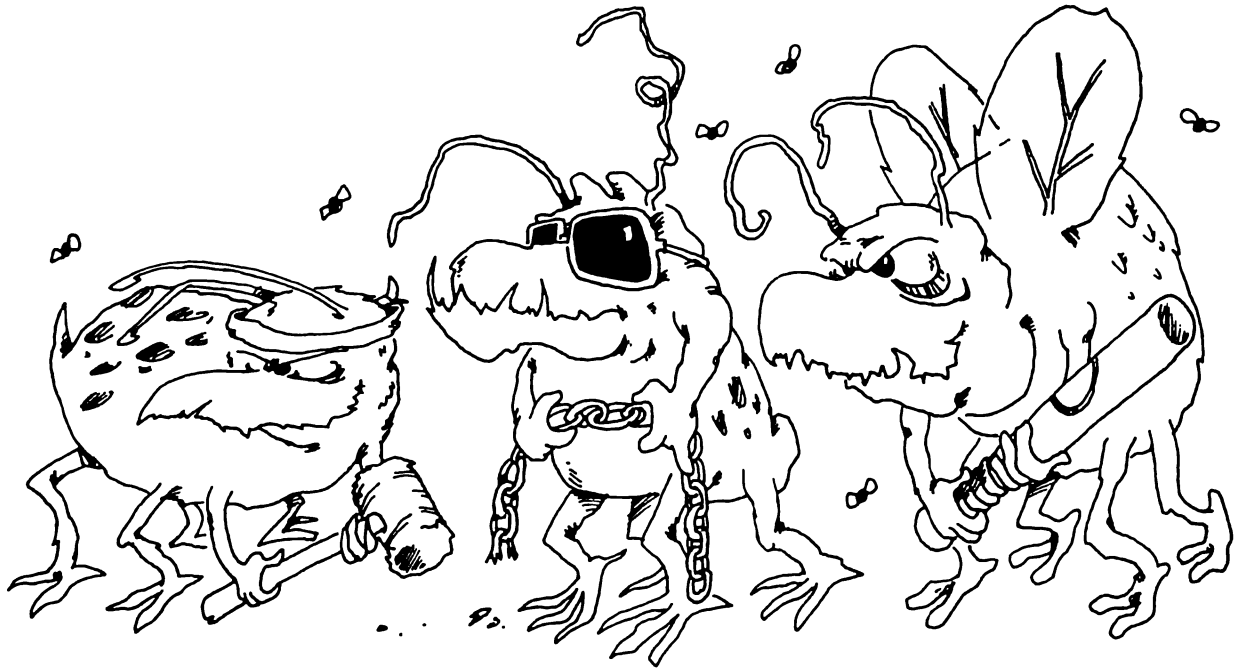
CONT	GOTO XX
These two ways use the values in the variable boxes left over from the last time you ran.	

CONT	starts at the line where the BREAK occurred.
GOTO XX	starts at line XX

RUN	RUN XX
These two ways throw away all the variable boxes made the last time, then execute the program.	

RUN	starts at the first line of the program
RUN XX	starts at line XX

CONT can only restart a program that was stopped with a break from a STOP or CTRL-C. But RUN, RUN XX and GOTO XX can also start a new program.



## **DEBUGGING**

Little errors in your program are called “bugs.”

If your program doesn’t run right, do these four things:

1. If the computer printed an **ERROR MESSAGE**, it tells what line it stopped on. Careful, the mistake may really be in another line!
2. If the computer just keeps running but doesn’t do the right thing, stop it and put some **PRINT** lines in that will tell what is happening.
3. Or you can put **STOP** commands in the program.
4. If the program runs so fast that you can’t tell what is happening, put in some delay loops to slow it down.

After you have fixed the program, take the **PRINT** lines, the **STOP**’s and the delay loops out of the program.

### **Assignment 33:**

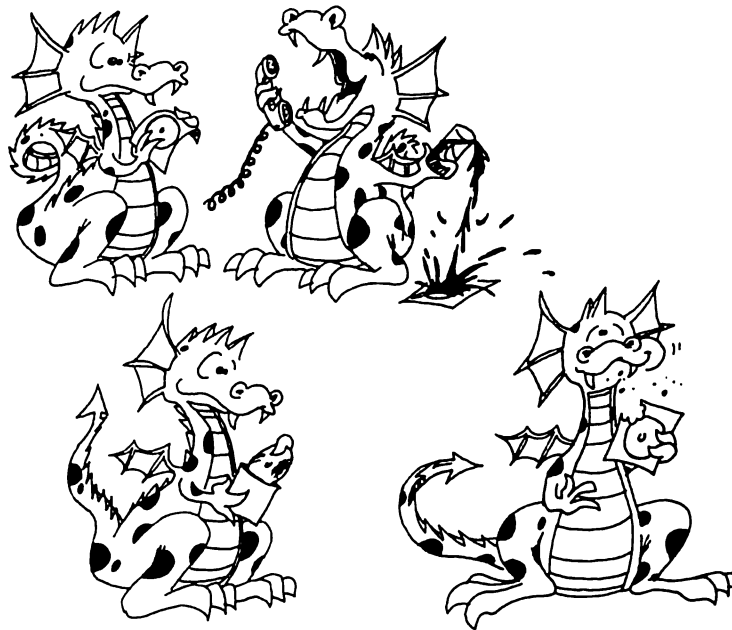
1. Go back to the **SNAKE** program and fix up some of the bugs. For example, the program “crashes” when the snake hits a wall. Add “food” for the snake. Add score keeping. Let the game end if the snake touches a wall.
2. Go back and fix up some other program that you have written.

## APPENDIX A DISK USAGE

### DISK CARE

The student should be instructed on care of diskettes at this time. Especially:

1. *Do not touch the brown or gray magnetic disk though the oblong holes.*
2. *Do not bend the diskette.*
3. *Insert and remove the disk carefully from the drive.*
4. *Always put the disk back in its protective cover after use.*
5. *Do not spill food or drink on the diskette. In fact, it is better to not snack while at the computer.*



### PREPARING A DISK FOR THE STUDENT

Enter this program:

```
10 REM === HELLO ===
20:
30 REM GREETING PROGRAM FOR
40:
50 REM "KIDS AND THE APPLE"
55:
60 HOME:PRINT
70 PRINT "student's name'S DISK"
75 PRINT CHR$(4); "CATALOG"
77 PRINT
80 PRINT "ENTERING EDIT MODE"
85 NEW
```

Where it says "student's name" put the name of your student. This HELLO program will print the student's name, then the catalog of the disk and finally erase itself from memory so that the student will not accidentally mix this HELLO program into his/her own program.

After entering the above program, put a new disk in drive 1 and enter:

```
INIT HELLO
```

The disk drive will grind for a long time, preparing the disk for files.

Then put in the DOS 3.3 SYSTEM MASTER and

```
LOAD COLOR DEMOSOFT
```

Put the student's disk back in and

```
SAVE COLOR DEMOSOFT
```

It may be a good idea to:

```
LOCK HELLO  
LOCK COLOR DEMOSOFT
```

so that they will not accidentally be erased from the disk. The locked status of the files is indicated by an "\*" before the entry in the catalog.

If you want to modify the HELLO program in the future:

```
UNLOCK HELLO  
LOAD HELLO
```

Make your modifications. Then:

```
SAVE HELLO  
LOCK HELLO
```

## APPENDIX B SAVING TO TAPE

If your computer does not have a disk drive, you will use a tape recorder to save your programs.

There are two steps in saving programs on cassette tape:

First, you must be sure that the volume and tone controls are set correctly on the tape recorder.

Then you must enter the correct commands into the computer.

### SETTING THE TAPE RECORDER CONTROLS

The first time a tape recorder is used with the Apple II, the correct setting for the volume control must be found. If this has already been done, the instructor will write the settings down on the lines below.

Otherwise, the instructor will follow the directions given in the Apple II BASIC Programming Manual for finding the correct settings and then write them down here.

---

---

---

---

### SAVING A PROGRAM TO CASSETTE TAPE

The easiest way is to put only one program on each cassette. The best kind of cassettes to use are the 10 minute tape cassettes made especially for recording data. They are quite inexpensive.

### FOLLOW THESE STEPS

1. List your program to make sure it is still there.
2. Rewind the tape.
3. Type SAVE but do not push the RETURN key yet!
4. Push the RECORD and PLAY buttons at the same time on the recorder to start it.
5. Push the RETURN key on the computer.

From here on, the computer does everything automatically. You will see and hear these things:

1. The cursor will disappear
2. While you count to 15 slowly, the computer puts a steady tone on the tape, but you do not hear this happening.
3. Then the computer will peep. It is starting to record the program now.
4. When the computer is done recording, it will peep again, and the flashing cursor will reappear.

The computer THINKS it put a program on the tape! To find out if the program really IS on tape, LOAD the program back into the computer.

### **LOADING A PROGRAM FROM TAPE INTO THE COMPUTER MEMORY**

1. Look to see if the recorder controls are set properly.
2. Rewind the tape.
3. Enter LOAD.
4. Start the recorder by pressing the PLAY key.

From here on the computer does things automatically. You should see and hear these things:

1. The cursor will disappear. The computer is listening for the steady tone. You will not hear it yourself.
2. About 15 seconds later the computer will peep. It has come to the end of the steady tone and started to load the program into memory.
3. When the computer has finished loading the program, it will peep and the flashing cursor will reappear. It takes about 10 seconds between peeps to load a short program and a few minutes to load a long one.



## ERRORS IN LOADING TAPE

If the computer has trouble loading the program it will print an error message and may beep too. If you see one of these messages:

```
*** SYNTAX ERR
ERR
*** MEM FULL ERR
ERR*** MEM FULL ERR
```

ask for help from the instructor or read the Apple II BASIC Programming Manual.

## PLAN AHEAD

Sometimes things go wrong when you try to save a program on cassette. It is very discouraging to lose a program that you worked hard to write! You want to be sure that saving to tape will work BEFORE you write the program.

*So when you sit down at the computer, the first thing you should do is check to see if the tape system is working ok. Do this:*

1. Load a short program into the computer from tape.
2. Save it on another cassette.
3. Then erase memory with a NEW command.
4. Read in the program you just saved to make sure it really was saved ok.

## SORRY, NO FILES

You can't use file names with tape storage like you can with disk storage.

wrong (with tape)	LOAD SNAKE
wrong (with tape)	SAVE SNAKE
right	LOAD
right	SAVE

## APPENDIX C RESERVED WORDS IN APPLESOFT

&				
AND	ASC	AT	ATN	
CALL	CHR\$ COS	CLEAR	COLOR =	CONT
DATA	DEF	DEL	DIM	DRAW
END	EXP			
FLASH	FN	FOR	FRE	
GET	GOSUB	GOTO	GR	
HCOLOR =	HGR HOME	HGR2 HPLOT	HIMEM: HTAB	HLIN
IF	IN#	INPUT	INT	INVERSE
LEFT\$	LEN LOG	LET LOMEM:	LIST	LOAD
MID\$				
NEW	NEXT	NORMAL	NOT	NOTRACE
ON	ONERR	OR		
PDL	PEEK POS	PLOT PRINT	POKE PR#	POP
READ	RECALL RETURN RUN	REM RIGHT\$	RESTORE RND	RESUME ROT =
SAVE	SCALE = SIN STEP	SCRN ( SPC ( STOP	SGN SPEED = STORE	SHLOAD SQR STR\$
TAB (	TAN TRACE	TEXT	THEN	TO
USR				
VAL	VLIN	VTAB		
WAIT				
XPLOT	XDRAW			

## ANSWERS TO ASSIGNMENTS

### ASSIGNMENT 1-3

```
10 HOME
20 PRINT "HI THERE,"
30 PRINT "APPLE COMPUTER!"
```

#### 2-1

```
10 HOME
20 FLASH
25 PRINT "MINDA"
30 INVERSE
35 PRINT "ANNE"
40 NORMAL
45 PRINT "CARLSON"
```

#### 2-2

```
10 HOME
20 FLASH
22 PRINT CHR$ (7)
25 PRINT "MINDA"
30 INVERSE
32 PRINT CHR$ (7)
35 PRINT "ANNE"
40 NORMAL
42 PRINT CHR$ (7)
45 PRINT "CARLSON"
```

#### 3-5

```
15 HOME
16 PRINT
17 PRINT
20 PRINT "--- ---"
25 PRINT "  0  "
27 PRINT  CHR$ (7)
30 PRINT
32 PRINT
34 PRINT
36 PRINT "      --- ---"
38 PRINT "          0  "
40 PRINT  CHR$ (7)
50 PRINT
52 PRINT
54 PRINT
56 PRINT "          ---0---"
58 PRINT "              0  "
60 PRINT  CHR$ (7)
```

#### 4-2

```

10 HOME
20 PRINT
22 PRINT
24 PRINT
26 PRINT
30 PRINT "    00    00    "
32 PRINT "    00    00    "
34 PRINT
36 PRINT
38 PRINT
40 PRINT
42 PRINT "    *          *    "
44 PRINT "    *          *    "
46 PRINT "    **        **    "
48 PRINT "    *****    "

```

#### 5-1

```

10 HOME
20 PRINT
22 PRINT
24 PRINT
30 PRINT " HELLO. WHAT IS YOUR NAME?"
35 INPUT N$
37 HOME
38 PRINT
40 PRINT "WELL,"
41 PRINT
42 PRINT N$
45 PRINT
50 PRINT "IT IS SILLY TO TALK TO COMPUTERS."

```

#### 5-2

```

10 HOME
20 PRINT "WHAT IS YOUR FAVORITE COLOR?"
25 INPUT C$
27 PRINT
30 PRINT "I PUT THAT IN BOX C$"
32 PRINT
35 PRINT "NOW YOUR FAVORITE ANIMAL?"
40 INPUT C$
42 PPRINT
45 PRINT "I PUT THAT IN BOX C$ TOO"
47 PRINT
50 PRINT "NOW LET'S PRINT WHAT IS IN BOX C$"
52 PRINT
55 PRINT "IT IS:"
57 PRINT
60 PRINT C$

```

## 6-1

```

2  REM  BY DAN CLARK, AGE 10
10 HOME
20 PRINT "NAME A MUSICAL GROUP "
30 INPUT A$
40 PRINT
50 PRINT "NAME ONE OF THEIR SONGS "
60 PRINT
70 INPUT B$
75 PRINT
80 PRINT A$;" PLAYS ";B$

```

## 6-2

```

10 HOME
20 PRINT "GIVE ME THE NAME OF A MUSICAL GROUP"
22 INPUT G$
25 PRINT
30 PRINT "WHAT IS ONE OF THEIR TUNES?"
35 INPUT T$
40 PRINT
42 PRINT
50 PRINT G$;
52 PRINT " PLAYS ";
54 PRINT T$

```

## 6-3

```

10 HOME
20 PRINT "YOUR NAME?"
25 INPUT N$
30 INVERSE
40 SPEED= 1
42 PRINT
44 PRINT
46 PRINT
50 PRINT N$
60 NORMAL
70 SPEED= 255

```

## 7-2

```

10 HOME
15 PRINT
17 PRINT
20 PRINT " HOW IS THE WEATHER?"
25 INPUT W$
28 PRINT
30 PRINT "AND HOW DO YOU FEEL?"
35 INPUT F$
37 PRINT
40 PRINT "YOU MEAN:"
42 PRINT
45 S$ = W$ + " AND " + F$
50 PRINT S$;"?"

```

# 8-3

```

10 HOME
15 LET M$ = "MINDA"
20 LET Y$ = "KAREN"
30 PRINT M$
35 SPEED= 1
40 PRINT Y$
50 GOTO 30

```

# 9A-2

```

10 REM BOYS AND GIRLS
20 HOME
22 PRINT
23 PRINT
25 PRINT "ARE YOU A BOY OR A GIRL?"
30 INPUT A$
32 PRINT
33 PRINT
34 PRINT
35 IF A$ = "BOY" THEN PRINT "SNIPS AND SNAILS"
40 IF A$ = "GIRL" THEN PRINT "SUGAR AND SPICE"

```

# 9B-1

```

2 REM PIZZA
3 REM BY CHRIS CLARK, JR. AGE 14 GOING ON
  (YOU FIGURE IT OUT)
4 HOME
5 SPEED= 100
6 PRINT "HALLO, AY AM MARIO, YOUR PIZZA MAN."
7 PRINT "JUST TELL ME ZE GORY DETAILS AND I'LL"
8 PRINT "DO ZE REST,"
9 PRINT
10 PRINT "WHAT SIZE SHOULD ZIS PIZZA BE
   (S/M/L)?"
20 INPUT S$
30 IF S$ = "S" THEN PRINT "ON A DIET? HO HO!"
31 PRINT
33 IF S$ = "M" THEN PRINT "GOOD CHOICE-NOT TOO
   BIG, BUT FILLING!"
35 PRINT
38 IF S$ = "L" THEN PRINT "YOU MUST HAVE A BIG
   BUNCH AT HOME!"
39 PRINT
40 PRINT "NOW, YOU WANT DOUBLE CHEES ON ZIS
   (Y/N)?"
42 INPUT CH$
45 REM ETC.
50 REM MUSHROOMS, ETC.
60 REM ANCHOVIES, ETC.
80 REM PEPPERS, ETC.
90 REM MEAT, ETC.

```

```

150 INVERSE
151 PRINT "HOKAY, HERE IS YOUR PIZZA!"
152 NORMAL
153 PRINT
154 IF S$ = "S" THEN PRINT "WAN SMALL PIZZA WITH
    ";
155 REM ETC.
160 IF BASE$ = "P" THEN PRINT "PEPPERONI"
165 REM ETC., ETC.
238 PRINT
240 FOR J = 1 TO 2000
242 NEXT J
244 SPEED= 255

```

#### 9B-2

```

10 REM === COLOR GUESSING GAME ===
20 HOME
22 PRINT
23 PRINT
24 PRINT
25 PRINT "PLAYER 2 TURN YOUR BACK"
27 PRINT
30 PRINT "PLAYER 1 ENTER A COLOR"
35 INPUT C$
40 HOME
42 PRINT
43 PRINT
44 PRINT
50 PRINT "PLAYER 2 TURN AROUND AND GUESS"
52 PRINT
53 PRINT
54 PRINT
55 INPUT G$
60 IF G$ < > C$ THEN PRINT "WRONG!"
65 IF G$ = C$ THEN PRINT "RIGHT"
67 PRINT
70 GOTO 55

```

#### 10-1

```

10 REM BIRTH YEAR
20 HOME
30 PRINT "HOW OLD ARE YOU?"
32 PRINT
34 INPUT A
36 PRINT
40 PRINT "AND WHAT YEAR IS IT NOW?"
50 INPUT Y
52 B = Y - A
55 PRINT "HAS YOUR BIRTHDAY COME YET THIS YEAR?"
56 PRINT "<Y/N> "
60 INPUT Y$
65 IF Y$ = "N" THEN B = B - 1
67 PRINT
70 PRINT "YOU WERE BORN IN ";B;","

```

### 10-2

```

10  REM MULTIPLICATION
20  HOME
22  PRINT
23  PRINT
24  PRINT
30  PRINT "GIVE ME A NUMBER"
35  INPUT A
37  PRINT
38  PRINT
40  PRINT "GIVE ME ANOTHER"
45  INPUT B
48  C = A * B
50  PRINT
52  PRINT
60  PRINT "THEIR PRODUCT IS ";C

```

### 10-3

```

10  REM CLOCK
20  HOME
30  PRINT "CLOCK PROGRAM"
40  INPUT "PRESENT TIME: HR,MIN,SEC? ";H,M,S
50  MUL = 1018
60  FOR I = 1 TO MUL: NEXT
70  S = S + 1
80  PRINT H:" " TAB ( 5)M:" " TAB ( 10)S
90  GOTO 60
100 REM NEEDS "IF. . ." TO TURN MIN,HR HANDS

```



11A-1

```

10 HOME
12 PRINT
13 PRINT
14 PRINT
15 PRINT "THIS IS A PARTY GAME"
16 PRINT
17 PRINT
20 PRINT "WHAT IS YOUR NAME? "
22 PRINT
24 PRINT
25 INPUT F$
27 PRINT
30 PRINT "PLEASE TURN YOUR BACK, ";F$
31 FOR I=1 TO 5000:NEXT I
32 HOME
34 PRINT "SOMEONE PRINT IN A NICKNAME ";
36 INPUT N$
40 HOME
42 PRINT
44 PRINT
50 PRINT CHR$ (7):REM BELL
55 PRINT F$; TAB ( 10);"IS CALLED `";N$;"`"
60 GOTO 12

```

```

      11A-2
10  REM !@**$ INSULTS **@!
12  HOME
14  PRINT
16  PRINT
20  PRINT "HEY YOU! WHAT IS YOUR NAME? "
21  PRINT
22  PRINT
23  INPUT N$
25  REM  DELAY LOOP
30  FOR T = 1 TO 2000
35  HOME
36  PRINT  CHR$ (7)
38  PRINT
40  PRINT  CHR$ (7)
42  PRINT
44  PRINT  CHR$ (7)
46  PRINT
48  PRINT  CHR$ (7)
50  PRINT "BAH!!!"
52  PRINT
55  PRINT  TAB( 10);N$
57  PRINT
60  PRINT  TAB( 15);"YOUR FATHER EATS LEEKS!!!"

```

```

      11B-1
2   REM "PATIENCE" BY DAN CLARK, AGE 10
10  HOME
15  SPEED= 2
18  PRINT  CHR$ (7)
20  PRINT "YOUR. . ."
22  PRINT
23  FOR I = 1 TO 2000: NEXT I
25  PRINT  CHR$ (7)
30  PRINT "    . . .WAIT'S. . ."
31  PRINT
32  FOR I = 1 TO 2000: NEXT I
35  PRINT  CHR$ (7)
40  PRINT "                ...OVER!!"
50  SPEED= 255

```

### 12B-3

```

10 REM I GOT YOUR NUMBER!
20 HOME
25 PRINT
26 PRINT
27 PRINT
30 PRINT "GIVE ME A NUMBER BETWEEN ZERO AND
    TEN."
35 PRINT
40 INPUT N
45 PRINT
46 PRINT
50 IF N = 0 THEN PRINT "I GOT PLENTY OF
    NOTHING!"
51 IF N = 1 THEN PRINT "I'M NUMBER ONE!"
52 IF N = 2 THEN PRINT "TWO IS COMPANY."
53 REM ETC.
70 IF N > 10 THEN GOTO 99
90 GOTO 25
99 PRINT "THAT'S ALL FOLKS!"

```

### 13-1

```

10 REM ** A PAIR OF DICE **
15 HOME
20 LET D1 = 1 + INT ( RND (8) * 6)
22 LET D2 = 1 + INT ( RND (8) * 6)
25 D = D1 + D2
30 PRINT "THE ROLL GAVE:"
32 PRINT
35 PRINT TAB( 10);"THE FIRST DIE " ;D1
40 PRINT TAB( 10);"THE SECOND " ;D2
45 PRINT TAB( 10);"THE DICE " ;D
47 PRINT
48 PRINT
50 PRINT "AGAIN? Y/N";
55 INPUT Y$
57 PRINT
58 PRINT
60 IF Y$ = "Y" THEN GOTO 20

```

## 13-2

```

10  REM PAPER, SCISSORS AND ROCK
12  HOME
13  PRINT
14  PRINT
15  PRINT
16  PRINT "PLAY THE "
17  PRINT
18  PRINT
19  PRINT TAB( 5)"P A P E R"
20  PRINT TAB( 15)"S C I S S O R S"
21  PRINT TAB( 31)"R O C K"
22  PRINT
23  PRINT "GAME AGAINST THE COMPUTER"
24  PRINT
25  PRINT "PRESS 'RESET' KEY TO END GAME"
26  PRINT
27  PRINT "ENTER YOUR CHOICE: <P,S,R> "
29  REM  COMPUTER CHOOSES
30  C = INT ( RND (8) * 3) + 1
32  IF C = 1 THEN C$ = "P"
33  IF C = 1 THEN C$ = "S"
34  IF C = 1 THEN C$ = "R"
35  REM C$ IS THE COMPUTERS CHOICE
37  INPUT Y$
38  REM  Y$ IS YOUR CHOICE
39  :
45  REM  IS THERE A TIE?
46  :
50  IF C$ < > Y$ THEN GOTO 60
52  REM  IF C$=Y$ THERE IS A TIE
55  PRINT "  TIE"
57  GOTO 30
59  :
60  REM  NO TIE, WHO WINS?
61  :
62  IF C$ = "P" THEN IF Y$ = "S" THEN GOTO 70
63  IF C$ = "S" THEN IF Y$ = "R" THEN GOTO 70
64  IF C$ = "R" THEN IF Y$ = "P" THEN GOTO 70
65  REM  COMPUTER WINS
66  PRINT "                COMPUTER WINS"
69  GOTO 30
70  REM  YOU WIN
72  PRINT "                YOU WIN"
79  GOTO 30
90  REM  END THE GAME BY PRESSING
92  REM  THE "RESET" KEY

```

```

        OUTLINE 15-2
10  REM  !!! VACATION !!!
12  HOME
20  REM  HEADING
30  REM  INSTRUCTIONS
50  REM  CHOICE OF VACATIONS
90  REM  ENDING OF PROGRAM

```

```

        PROGRAM 15-2
10  REM  !!! VACATION !!!
12  HOME
13  PRINT
14  PRINT
15  PRINT
20  REM  HEADING
21  PRINT "VACATION CHOOSING PROGRAM"
22  PRINT
23  PRINT "PICKS YOUR VACATION BY THE"
24  PRINT "AMOUNT YOU WANT TO SPEND"
25  PRINT
30  REM  INSTRUCTIONS
31  PRINT "ENTER THE AMOUNT IN DOLLARS THAT "
32  PRINT "YOU CAN SPEND."
33  PRINT
35  REM  GET DOLLAR AMOUNT
37  INPUT D
40  M$ = "FLIP PENNIES WITH YOUR KID BROTHER."
43  P$ = "SPEND THE AFTERNOON IN BEAUTIFUL HOG
    WALLOW, MICH."
45  Q$ = "ENTER A PICKLE EATING CONTEST IN
    SCRATCHY BACK, TENN."
47  REM  ETC.
49  REM  ETC.
58  Z$ = "BUY A COSY YACHT AND CRUISE THE
    CARIBBEAN SEA."
70  IF D < 0.49 THEN PRINT M$: GOTO 90
72  IF D < 0.99 THEN PRINT P$: GOTO 90
74  REM  ETC.
76  REM  ETC.
86  IF D < 1000000 THEN PRINT Z$: GOTO 90
88  PRINT "TREAT YOUR WHOLE SCHOOL TO A 'ROUND
    THE WORLD TRIP!"
90  REM  ENDING OF PROGRAM

```

## 16-1

```

10  REM  RANDOM NAME
20  HOME
30  INPUT "YOUR NAME? " ; N$
40  X = RND (8) * 35
50  Y = 1 + RND (8) * 19
60  HTAB (X)
70  VTAB (Y)
80  PRINT N$
90  GOTO 40
99  END

```

## 16-2

```

10  REM JUMPING "HERE"
20  HOME
30  X = RND (8) * 35 + 1
35  Y = RND (8) * 22 + 1
50  VTAB (Y)
60  HTAB (X)
70  PRINT "HERE";
75  FOR I = 1 TO 500: NEXT
80  HTAB (X)
90  PRINT "      "
999 GOTO 30

```

## 17A-1

```

10  REM  COUNTING BY FIVES
15  SPEED= 1
20  FOR I = 5 TO 100 STEP 5
30  PRINT I
40  NEXT I
99  SPEED= 255

```

## 17B-2

```

10  HOME
20  REM  SLIPPING NAME
25  N$ = "BRIAN"
27  SPEED= 200
30  FOR I = 1 TO 15
35  PRINT TAB( I * 2) ; N$
40  NEXT I
99  SPEED= 255

```

7B-3

```

10  REM  CLIMBING NAME
12  HOME
15  N$ = "STANLAUS MAZURSKI"
17  SPEED= 225
20  FOR I = 23 TO 1 STEP - 1
25  VTAB I
27  HTAB I
30  PRINT N$
40  NEXT
90  HOME
99  SPEED= 255

```

17B-4

```

10  REM +++ FRIENDS NAMES +++
12  HOME
15  SPEED= 100
20  INPUT "NAME? ";N$
22  PRINT
25  INPUT "OTHER NAME? ";M$
30  FOR I = 1 TO 5
35  FLASH
36  PRINT
40  PRINT N$
45  INVERSE
46  PRINT
50  PRINT M$
60  NEXT I
98  SPEED= 255
99  NORMAL

```

19-2

```

10  REM  SLIDING NAME
12  HOME
20  INPUT "YOUR NAME? ";N$
25  VTAB 15: HTAB 1
30  SPEED= 225
40  FOR I = 1 TO 30
45  N$ = " " + N$
50  PRINT N$;
55  HTAB 1
60  NEXT I
99  SPEED= 255

```

## 19-3

```

10  REM  FALLING FRIEND
12  HOME
20  INPUT "YOUR FRIEND'S NAME: ";F$
25  B$ = "
29  HOME
30  FOR I = 1 TO 23
31  PRINT B$
32  HTAB 15
34  VTAB I
40  PRINT F$;
42  HTAB 15
45  FOR T = 1 TO 100: NEXT T
49  NEXT I

```

## 19-4

```

10  REM  CROSSING FRIENDS
12  HOME
20  INPUT "YOUR NAME ";Y$
22  PRINT
24  INPUT "YOUR FRIEND'S NAME ";F$
26  B$ = "      "
28  HOME
35  :
36  REM  MAIN LOOP
37  :
40  FOR I = 1 TO 23
45  IF I > 16 THEN GOTO 60
50  HTAB 2 * I
51  VTAB 12
53  PRINT B$;
55  HTAB 2 * I + 2
57  PRINT Y$;
60  HTAB 12: VTAB I
65  PRINT B$;
70  HTAB 12: VTAB I + 1
75  PRINT F$;
90  NEXT I
91  :
92  REM  DELAY AT END
93  :
95  FOR T = 1 TO 2000: NEXT T
99  HOME

```



# 21-2

```

100 REM MAKE A STAR
105 GR
110 COLOR= 13
120 FOR I = 15 TO 25
125 PLOT 20,I
130 PLOT I,20
140 NEXT I
148 COLOR= 11
150 FOR I = 16 TO 24
155 PLOT I,I
158 J = 40 - I
160 PLOT J,I
170 NEXT I
180 FOR T = 1 TO 3000: NEXT T
195 TEXT
197 HOME

```

# 22-1

```

10 REM DRAW A BOX
12 HOME
13 PRINT : PRINT : PRINT
20 PRINT "WHAT COLOR SQUARE? <1-15> "
22 INPUT C
30 GR : COLOR= C
40 HLIN 10,30 AT 10
42 HLIN 10,30 AT 39
44 VLIN 10,39 AT 10
46 VLIN 10,39 AT 30
60 FOR T = 1 TO 5000: NEXT T
95 TEXT
97 HOME

```

```

21 - 3
100 REM SINBAD'S MAGIC CARPET
101 :
102 GOTO 1000
200 :
201 REM MAIN LOOP
202 :
205 X = 1
206 A = RND (8):B = RND (8)
207 C = RND (8)
210 FOR I = 0 TO 19: FOR J = 0 TO 19
212 K = I + J
214 COLOR= C(X)
215 PRINT C(X)
216 X = X + A * (I + 3 * B) / (J + 3) + C * (20 -
    I - J) / 53
217 REM CHANGE LINE 216 TO GET A VARIETY OF
    PATTERNS
218 IF X > NC * 1 THEN X = X - NC: GOTO 218
219 IF X < 1 THEN X = 1
230 PLOT I,K: PLOT K,I
232 PLOT 39 - I,K: PLOT K,39 - I
234 PLOT 39 - K,I: PLOT I,39 - K
236 PLOT 39 - I,39 - K: PLOT 39 - K,39 - I
290 NEXT J,I
999 END
1000 :
1001 REM <<< SINBAD'S MAGIC CARPET >>>
1002 :
1080 DIM C(15)
1090 GR
2000 :
2001 REM INPUT COLORS
2002 :
2010 INPUT " NUMBER OF COLORS IN THE RUG ";NC
2020 FOR I = 1 TO NC
2025 INPUT " COLOR? <0-15> ";C
2030 C(I) = C
2032 FOR J = 1 TO C:X = RND (8): NEXT J
2033 REM THIS CHANGES THE RANDOM NUMBER FROM
    ONE RUG TO THE NEXT
2035 NEXT I
2040 REM SOME OF THE PRETTIEST RUGS HAVE ONLY 2
    COLORS
9999 GOTO 200

```

## 22-2

```

10 REM DRAW A DIE
12 HOME
13 PRINT : PRINT : PRINT
20 PRINT "WHAT COLOR SQUARE? <1-15> "
22 INPUT C
30 GR : COLOR= C
40 HLIN 10,30 AT 10
42 HLIN 10,30 AT 39
44 VLIN 10,39 AT 10
46 VLIN 10,39 AT 30
100 :
101 REM ADD SPOTS
102 :
110 PRINT "HOW MANY SPOTS?"
115 INPUT NS
118 COLOR= 14
120 IF NS = 1 THEN PLOT 20, 25
125 IF NS = 2 THEN PLOT 13,14: PLOT 27,35
130 IF NS = 3 THEN PLOT 20,25: PLOT 13,14: PLOT 27,35
135 IF NS = 4 THEN PLOT 13,35: PLOT 27,14:NS = 2: GOTO 120
140 IF NS = 5 THEN PLOT 20,25:NS = 4: GOTO 120
145 IF NS = 6 THEN PLOT 20,14: PLOT 20,35:NS = 4: GOTO 120
160 FOR T = 1 TO 3000: NEXT T
195 TEXT
197 HOME

```

## 23-1

```

9 :
10 REM MENU MAKER
11 :
12 HOME
15 PRINT : PRINT : PRINT
19 :
20 PRINT "WHICH COLOR DO YOU LIKE?"
21 PRINT
22 PRINT " <M> MAGENTA
24 PRINT " <Y> YELLOW
26 PRINT " <G> GREEN
28 PRINT " <B> BLUE
29 PRINT
30 :
31 GET C$
32 :
35 IF C$ = "M" THEN C = 1
36 IF C$ = "Y" THEN C = 13
37 IF C$ = "G" THEN C = 12
38 IF C$ = "B" THEN C = 6
39 :
40 COLOR= C
50 :
51 REM MAKE A STAR AS IN THE PROBLEM 2 OF ASSIGNMENT 21
52 :

```

# A23-2

```

10  REM  SILLY SENTENCES
12  HOME
13  PRINT : PRINT : PRINT
14  PRINT "SILLY SENTENCES": PRINT : PRINT
15  PRINT "WANT INSTRUCTIONS? <Y/N>": PRINT
16  GET Y$: IF Y$ = "Y" THEN GOTO 100
18  PRINT : PRINT
20  PRINT "THE SUBJECT: (END WITH A PERIOD)": PRINT
22  GET L$
24  IF L$ = "." THEN GOTO 30
28  S$ = S$ + L$
29  GOTO 22
30  S$ = S$ + " ": REM ADD A SPACE AFTER THE LAST WORD
32  PRINT "THE VERB: (END WITH A PERIOD)": PRINT
34  GET L$
36  IF L$ = "." THEN GOTO 42
38  S$ = S$ + L$
40  GOTO 34
42  S$ = S$ + " "
50  PRINT "THE OBJECT: (END WITH A PERIOD)": PRINT
52  GET L$
54  IF L$ = "." THEN GOTO 70
56  S$ = S$ + L$
58  GOTO 52
70  S$ = S$ + "."
80  PRINT
85  PRINT S$
99  END
100 :
101 REM  INSTRUCTINS
102 :
104 HOME
110 PRINT "THREE PLAYERS ENTER PARTS OF A SENTENCE": PRINT
115 PRINT "NO PLAYER CAN SEE WHAT THE OTHERS ENTER": PRINT
120 PRINT "THE FIRST ENTERS THE SUBJECT,": PRINT
122 PRINT "    (THE PERSON DOING SOMETHING)": PRINT
130 PRINT "THE SECOND ENTERS THE VERB": PRINT
132 PRINT "    (THE ACTION WORD)": PRINT
140 PRINT "THE THIRD ENTERS THE OBJECT": PRINT
142 PRINT "    (THE PERSON OR THING TO WHOM ": PRINT
144 PRINT "    THE ACTION IS DONE)": PRINT
148 FOR T = 1 TO 5000: NEXT T
150 GOTO 18

```

25-2

```

10  REM  APPLE TREE
12  HOME : GR
20  REM  MAIN LOOP
25  GOSUB 200: REM  SKY
30  GOSUB 300: REM  MAKE TREE
40  GOSUB 400: REM  MAKE APPLES
50  GOSUB 500: REM  APPLES FALL
99  END
200  REM
201  REM  BACKGROUND
202  REM
205  COLOR= 6: REM  MAKE APPLES
210  FOR I = 0 TO 39
215  HLIN 0,39 AT I
220  NEXT I
300  REM
301  REM  MAKE TREE
302  REM
310  REM  GROUND
312  COLOR= 10: REM  GREY
314  HLIN 0,39 AT 39
315  HLIN 0,39 AT 38
320  REM  TRUNK
322  COLOR= 8: REM  DARK GREEN
324  VLIN 25,38 AT 19
326  VLIN 25,38 AT 20
330  REM  GREEN PART
332  COLOR= 4
333  HLIN 17,21 AT 25
334  HLIN 15,25 AT 24
335  HLIN 13,27 AT 23
336  HLIN 10,30 AT 22
337  HLIN 9,31 AT 21
338  HLIN 8,32 AT 20
339  HLIN 9,32 AT 19
340  HLIN 8,31 AT 18
341  HLIN 10,29 AT 17
342  HLIN 11,29 AT 16
343  HLIN 13,26 AT 15
344  HLIN 12,26 AT 14
345  HLIN 14,24 AT 13
346  HLIN 16,23 AT 12
347  HLIN 17,21 AT 11
390  RETURN

```

```

400 REM
401 REM MAKE APPLES
402 REM
405 COLOR= 9: REM ORANGE
408 FOR K = 1 TO 40
409 REM PUT APPLES IN A RECTANGLE
410 X = INT ( RND (8) * 28) + 8
415 Y = INT ( RND (8) * 25) + 10
420 L = SCRN( X,Y)
424 REM ONLY PUT APPLES ON GREEN TREE
425 IF L = 4 THEN PLOT X,Y
430 NEXT K
490 RETURN
500 REM
501 REM FALLING APPLES
502 REM
509 REM LOOK IN RECTANGLE FOR APPLES
510 FOR K = 1 TO 1000
515 X = INT ( RND (8) * 28) + 8
520 Y = INT ( RND (8) * 25) + 10
530 L = SCRN( X,Y)
534 REM IF FIND ORANGE APPLE, DROP IT
535 IF L = 9 THEN GOSUB 600
540 NEXT K
590 RETURN
600 REM
601 REM DROP THE APPLE
602 REM
610 LL = 4: REM GREEN
620 COLOR= LL: PLOT X, Y: REM COVER EMPTY SPOT
625 Y = Y + 1: REM LOOK BELOW APPLE
630 L = SCRN( X,Y)
634 REM IF SEE GROUND, PLOT APPLE AND RETURN
635 IF L = 10 THEN COLOR= 9: PLOT X,Y - 1: RETURN
640 LL = L: REM COLOR TO COVER EMPTY SPOT
645 COLOR= 9: PLOT X,Y: REM APPLE ONE SQUARE LOWER
690 GOTO 620

```

## 26-1

```

10 REM CIPHER MAKER
12 HOME
13 PRINT
14 PRINT
20 PRINT "CODE MAKING PROGRAM"
21 PRINT
22 PRINT
30 PRINT "ENTER A SENTENCE FOR CODING:"
31 PRINT
32 INPUT S$
33 PRINT
35 L = LEN (S$)
40 FOR I = 1 TO L STEP 2
45 P$ = MID$ (S$,I,2)
50 Q$ = RIGHT$ (P$,1) + LEFT$ (P$,1)
55 L$ = L$ + Q$
60 NEXT I
65 PRINT
66 PRINT "HERE IS THE CODED SENTENCE:"
67 PRINT
70 PRINT L$

```

## 26-2

```

10 REM QUESTION ANSWERER
12 HOME
13 PRINT : PRINT : PRINT
20 PRINT "ENTER A QUESTION"
21 PRINT
25 INPUT Q$
28 L = LEN (Q$)
30 :
31 REM TAKE OFF THE QUESTION MARK
32 :
33 Q$ = LEFT$ (Q$,L - 1) + ","
34 PRINT
35 :
36 REM LOOK FOR THE END OF THE FIRST WORD
38 :
39 FOR I = 1 TO L
40 C$ = MID$ (Q$,I,1)
45 IF C$ = " " THEN S1 = I:I = L
46 NEXT I
47 :
48 REM LOOK FOR THE END OF THE SECOND WORD
49 :
50 FOR I = S1 + 1 TO L

```

```

52 C$ = MID$ (Q$,I,1)
54 IF C$ = " " THEN S2 = I:I = L
56 NEXT I
57 :
58 REM TURN THE WORDS AROUND
59 :
60 S$ = MID$ (Q$,S+1 + 1,S2 - S1)
62 V$ = LEFT$ (Q$,S1)
65 PRINT S$ + V$ + RIGHT$ (Q$,L - S2)

```

### 26-3

```

10 REM *** PIG LATIN ***
12 HOME
13 PRINT : PRINT : PRINT
20 PRINT "THIS IS A PIG LATIN PROGRAM"
21 PRINT
25 PRINT
30 INPUT "GIVE ME A WORD: ";W$
32 PRINT : PRINT
40 F$ = LEFT$ (W$,1)
42 E$ = F$ + "AY"
44 IF F$ = "A" THEN E$ = "LAY"
46 IF F$ = "E" THEN E$ = "LEE"
48 IF F$ = "I" THEN E$ = "LIE"
50 IF F$ = "O" THEN E$ = "LO"
52 IF F$ = "U" THEN E$ = "LU"
70 L$ = RIGHT$ (W$, LEN (W$) - 1) + E$
80 PRINT L$
85 PRINT : PRINT
90 GOTO 30

```

### 27-1

```

10 REM BACKWARD ADDED TO FORWARD
12 HOME
13 PRINT : PRINT : PRINT
20 INPUT "GIVE ME A NUMBER: ";N
30 N$ = STR$ (N)
35 L = LEN (N$)
40 FOR I = 1 TO L
45 B$ = B$ + MID$ (N$,L + 1 - I,1)
50 B = VAL (B$)
55 NEXT I
57 PRINT : PRINT : PRINT : PRINT : PRINT
60 PRINT " ";N
61 PRINT " +";B
65 PRINT " "; LEFT$ ("-----",L)
70 A = N + B
72 A$ = STR$ (A)
75 IF LEN (A$) = L THEN PRINT " ";A
76 IF LEN (A$) = L + 1 THEN PRINT " ";A
99 END

```



# 27-2

```

10  REM  MARCHING NUMBERS
12  HOME
13  PRINT : PRINT : PRINT
20  INPUT "GIVE ME A NUMBER ";N
22  B$ = " "
25  N$ = STR$ (N)
27  L = LEN (N$)
28  VTAB 15
31  :
33  REM  LOOP
34  :
40  FOR I = 1 TO 39 - L
45  HTAB I
50  PRINT " ";
55  HTAB I + 1
60  PRINT N$;
62  FOR T = 1 TO 200: NEXT T
65  N$ = RIGHT$ (N$,L - 1) + LEFT $ (N$,1)
70  NEXT I
80  :
82  REM  END
84  :
99  SPEED= 255

```

# 28-1

```

200  REM  SHOOT
202  REM  ADD TO SHOOT PROGRAM
210  FOR I = 38 TO 0 STEP - 1
230  COLOR= 12
240  PLOT X,I
250  COLOR= 0
260  PLOT X,I
290  NEXT I
299  RETURN

```

## 28-2

```
10  REM  *** SHOOTING STARS ***
20  HOME : GR
30  GOSUB 300
40  PRINT "USE PADDLE 0"
50  Y = 39
55  COLOR= 0
60  PLOT X,Y
65  X = PDL (0) * 39 / 255
67  COLOR= 7
70  PLOT X,Y
75  IF PEEK ( - 16287) > 127 THEN GOSUB 200
99  GOTO 55
200 REM SHOOT
210 FOR I = 38 TO 0 STEP - 1
212 CS = SCRN( X,I)
215 IF CS = 1 THEN PRINT CHR$(7);:CS = 0
217 COLOR= 12
220 PLOT X,I
230 COLOR= CS
240 PLOT X,I
280 NEXT I
299 RETURN
300 REM INITIALIZE STARS
310 FOR I = 1 TO 10
320 X = RND (8) * 40
330 Y = RND (8) * 20
340 COLOR= 10
342 IF RND (8) < .5 THEN COLOR= 1
345 PLOT X,Y
350 NEXT I
399 RETURN
```

## 29-1

```

10 REM ALPHABETICAL
12 HOME
13 PRINT : PRINT : PRINT
19 :
20 PRINT "THIS PROGRAM ARRANGES THE LETTERS"
21 PRINT "OF A WORD IN ALPHABETICAL ORDER."
22 :
23 PRINT
29 :
30 INPUT "GIVE ME A WORD: ";W$
31 :
32 PRINT
35 L = LEN (W$)
37 :
38 REM TEST LETTERS IN ALPHABET
39 :
40 FOR I = 65 TO 65 + 26
41 :
42 REM TO SEE IF IN WORD
43 :
45 FOR J = 1 TO L
50 G = ASC ( MID$ (W$,J,1))
55 IF G = I THEN H$ = H$ + CHR$ (G)
60 NEXT J: NEXT I
61 :
70 PRINT
75 PRINT "HERE IT IS IN ALPHABETICAL ORDER:"
76 PRINT
80 PRINT H$

```

## 29-2

```

10 REM   @@@ DOUBLE DUTCH @@@
11 :
12 HOME : PRINT : PRINT : PRINT : PRINT : PRINT
20 :
21 REM INPUT THE SENTENCE
22 :
25 PRINT "GIVE ME A SENTENCE:": PRINT
30 INPUT S$: PRINT
40 L = LEN (S$)
42 HOME : PRINT : PRINT : PRINT : PRINT : PRINT
46 :
47 REM CHECK EACH LETTER
48 :
50 FOR I = 1 TO L
52 L$ = MID$ (S$,I,1): REM GET A LETTER
60 IF L$ = "A" THEN GOTO 72
62 IF L$ = "E" THEN GOTO 72
64 IF L$ = "I" THEN GOTO 72
66 IF L$ = "O" THEN GOTO 72
68 IF L$ = "U" THEN GOTO 72

```

```

69 SS$ = SS$ + L$
72 NEXT I
73 :
74 REM PRINT THE SENTENCE IN DOUBLE DUTCH
75 :
76 PRINT "HERE IT IS IN DOUBLE DUTCH "
77 PRINT
80 PRINT SS$

```

### 29-3

```

10 REM ON. . GOTO SAMPLE
12 HOME
13 PRINT : PRINT : PRINT
20 :
21 REM MAKE A MENU
22 :
30 PRINT "MAKE YOUR CHOICE:"
31 PRINT : PRINT "<A> TAKE A NAP"
32 PRINT : PRINT "<B> EAT AN APPLE"
33 PRINT : PRINT "<C> CALL A FRIEND"
40 PRINT : INPUT X$: PRINT
41 X = ASC (X$) - 64
49 :
50 ON X GOTO 60,70,80
51 :
52 GOTO 30
59 :
60 PRINT "YOUR BED IS NOT MADE!"
61 END
70 PRINT "YOUR SISTER ATE THE LAST ONE!"
71 END
80 PRINT "YOUR FATHER IS ON THE PHONE!"
81 END

```

### 30-1

```

9 :
10 REM PHONE LIST KEEPER
11 REM RUN 1000 TO INITIALIZE PHONE LIST
13 D$ = CHR$ (4)
14 DIM NA$(20),TE$(20)
15 :
16 REM MENU
17 :
18 GOSUB 800
20 HOME
21 PRINT : PRINT : PRINT "PHONE LIST": PRINT
22 PRINT " <A> ADD NAMES AND NUMBERS": PRINT
24 PRINT " <L> LIST THE NAMES AND NUMBERS": PRINT
25 PRINT " <E> END THE PROGRAM": PRINT
26 PRINT " <I> INSTRUCTIONS": PRINT
27 GET Y$
28 IF Y$ = "A" THEN GOTO 100
30 IF Y$ = "L" THEN GOTO 300

```

```

31 IF Y$ = "E" THEN END
32 IF Y$ = "I" THEN GOSUB 900
35 GOTO 27
95 END
100 :
101 REM ADD NEW NAMES AND NUMBERS
102 :
105 HOME
110 INPUT "ENTER NAME: ";A$
115 PRINT
120 INPUT "ENTER NUMBER: ";T$
130 FOR I = 1 TO 20
132 IF NA$(I) = "" THEN Z = I:I = 20
134 NEXT I
140 NA$(Z) = A$:TE$(Z) = T$
190 GOSUB 500
193 HOME
195 GOTO 20
300 :
301 REM LOOK UP A NUMBER
302 :
305 HOME
310 FOR I = 1 TO 20
320 PRINT NA$(I); TAB(25);TE$(I)
325 IF NA$(I) = "" THEN I = 20
330 NEXT I
390 PRINT "PRESS <M> FOR RETURN TO THE MENU": GET A$
395 GOTO 20
500 :
501 REM PUT THE PHONE LIST ON DISK
502 :
515 PRINT D$;"OPEN PHONE LIST"
520 PRINT D$;"WRITE PHONE LIST"
530 FOR I = 1 TO Z+ 1
535 PRINT NA$(I)
536 PRINT TE$(I)
540 NEXT I
545 PRINT D$;"CLOSE PHONE LIST"
595 RETURN
800 :
801 REM LOAD PHONE LIST FROM DISK
802 :
810 PRINT D$;"OPEN PHONE LIST"
811 PRINT D$;"READ PHONE LIST"
815 FOR I = 1 TO 20
820 INPUT NA$(I)
821 INPUT TE$(I)
825 IF NA$(I) = "" THEN Z = I:I = 20
830 NEXT I
840 PRINT D$;"CLOSE PHONE LIST"
895 RETURN
900 REM
995 RETURN
1000 :

```

```

1001 REM INITIALIZE DISK FILE "PHONE LIST"
1002 :
1003 D$ = CHR$ (4)
1004 PRINT D$;"OPEN PHONE LIST"
1005 PRINT D$;"WRITE PHONE LIST"
1007 DIM NA$(20),TE$(20)
1010 FOR I = 1 TO 20
1015 PRINT NA$(I)
1016 PRINT TE$(I)
1020 NEXT I
1040 PRINT D$;"CLOSE PHONE LIST"
1095 END

```

31-3

```

2 :
10 REM "AIN'T GOT NO"
11 :
12 HOME
13 PRINT : PRINT : PRINT
20 :
21 PRINT "SEARCH FOR DOUBLE NEGATIVES": PRINT
22 :
30 PRINT "ENTER A SENTENCE:": PRINT
31 :
32 INPUT S$: PRINT
33 S$ = S$ + " ": REM END THE SENTENCE WITH A BLANK SPACE.
35 L = LEN (S$)
40 NM = 0: REM NUMBER OF NEGATIVE WORDS
42 S1 = 1:S2 = 1: REM START LETTER OF TWO WORDS
45 FOR I = 1 TO L
50 L$ = MID$ (S$,I,1): REM GET A LETTER
54 REM IS IT A SPACE?
55 IF L$ = " " THEN S1 = S2:S2 = I + 1: GOSUB 200
60 NEXT I
65 PRINT
70 IF NM = 2 THEN PRINT "THIS SENTENCE HAS A DOUBLE NEGATIVE."
72 IF NM = 0 THEN PRINT "THIS SENTENCE AIN'T GOT NO DOUBLE
NEGATIVE!"
74 IF NM = 4 THEN PRINT "THIS SENTENCE HAS TWO DOUBLE NEGATIVES
100 :
101 REM TEST THE PROGRAM ON THE SENTENCE:
102 :
103 REM I NEVER EAT NO JUNK FOOD!
104 :
198 END
199 :
200 REM IS THE WORD A NEGATIVE?
201 :
205 LW = S2 - S1 - 1: REM LENGTH OF THE WORD
210 W$ = MID$ (S$,S1,LW)
220 PRINT W$
232 IF W$ = "NO" OR W$ = "NOT" OR W$ = "DON'T" THEN NN = NN + 1
234 IF W$ = "NOTHING" OR W$ = "NEVER" OR W$ = "AIN'T" THEN
NN = NN + 1

```

```

240 REM ETC.
295 RETURN

      32 - 2
1 GOTO 1000: REM *** CODE- DECODE ***
2 :
100 REM MAIN LOOP
110 REM
113 GOSUB 400: REM GET PASSWORD
115 PRINT : PRINT "CODE OR DECODE? <C/D> ": GET Y$
120 IF Y $ = "C" THEN GOTO 500: REM CODE MESSAGE
130 IF Y $ = "D" THEN GOTO 600: REM DECODE MESSAGE
140 GOTO 115
199 END
399 :
400 REM GET PASSWORD AND FORM CODE ALPHABET
401 :
402 HOME : PRINT : PRINT
405 INPUT "INPUT PASSWORD ";PW$
406 REM REMOVE REPEATED LETTERS FROM THE PASSWORD
408 F$ = LEFT$ (PW$,1)
410 FOR I = 2 TO LEN (PW$)
411 L1$ = MID$ (PW$,I,1)
412 FOR J = 1 TO LEN (F$)
415 L2$ = MID$ (F$,J,1)
420 IF L1$ = L2$ THEN GOTO 430
421 NEXT J
422 F$ = F$ + L1$
430 NEXT I
432 PW$ = F$
433 PRINT : PRINT : "THE SHORTENED PASSWORD IS ";PW$
434 :
435 REM REMOVE PASSWORD LETTERS FROM THE ALPHABET
436 :
440 FOR J = 1 TO LEN (PW$):L2$ = MID$ (PW$,J,1)
441 IF L2$ = LEFT$ (A$,1) THEN A$ = MID$ (A$,2): GOTO 460
442 FOR I = 2 TO LEN (A$):L1$ = MID$ (A$,I,1)
445 IF L1$ = L2$ THEN A$ = LEFT$ (A$,I - 1) + MID$ (A$,I + 1)
455 NEXT I
460 NEXT J
461 :
462 REM FORM CODE ALPHABET
463 :
465 A$ = PW$ + A$
470 PRINT : PRINT "THE CIPHER ALPHABET IS ": PRINT : PRINT A$:
PRINT B$
498 RETURN

```

```

499 :
500 REM FORM A CODED MESSAGE
501 :
505 PRINT : PRINT "INPUT MESSAGE, END WITH '*' SIGN": PRINT
510 GET L$:L = ASC (L$)
515 IF L$ = "*" THEN GOTO 590
520 IF L < 65 OR L > 91 THEN P$ = P$ + L$: GOTO 540
530 P$ = P$ + MID$ (A$,L - 64,1)
540 PRINT L$;
589 GOTO 510
590 PRINT : PRINT P$
598 END
599 :
600 REM DECODE A MESSAGE
601 :
610 PRINT : PRINT "TYPE IN THE CODED MESSAGE"
612 PRINT "END THE MESSAGE WITH A '*' SIGN"
613 PRINT : PRINT
615 GET L$
617 IF L$ = "*" THEN GOTO 690
620 FOR I = 1 TO 26
625 IF L$ = MID$ (A$,I,1) THEN PRINT MID$ (B$,I,1);: GOTO
630 NEXT I
635 PRINT L$;
640 GOTO 615
690 END
1000 :
1005 REM *** C O D E - D E C O D E ***
1010 :
2015 A$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2020 B$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2999 GOTO 110

```



```

    SOME PROGRAMS BY STUDENTS
0  REM  EXTRA 4 JOHN AND MATT
1  REM  <<<<<FIRE!>>>>>
2  REM
3  REM    BY MATT AND JOHN O'MALIA
4  REM    AGES  11 AND 12
5  REM
6  HOME : PRINT "PRESS PADDLE BUTTON TO HIT SHIP"
7  PRINT "AIM AT TAIL SECTION"
8  FOR I = 1 TO 2500: NEXT I
9  FOR GO = 1 TO 30
10 HOME : SOUND = PEEK ( - 16336)
15 HTAB GO: PRINT "*"
25 HTAB GO: PRINT "#####>"
35 HTAB GO: PRINT "*"
45 Y = PDL (0) / 8.5
47 VTAB 23: HTAB Y: PRINT "^"
60 IF PEEK ( - 16287) > 127 THEN 100
65 NEXT GO
99 GOTO 160
100 FOR AR = 22 TO 1 STEP - 1
105 VTAB AR: HTAB Y: PRINT "^"
120 FOR T = 1 TO 10: NEXT T
145 NEXT AR
147 Y = INT (Y) - 1
149 VTAB 15
150 PRINT "HE WAS AT ";GO;" YOU WERE AT ";Y
160 VTAB 17: PRINT "PLAY AGAIN? ";
170 GET AG$: IF AG$ = "Y" THEN RUN
180 END

```

```

2  REM  GREETING PROGRAM
3  REM  BY DAN CLARK, AGE 10
10  INPUT "WHAT IS YOUR NAME? ";N$
15  PRINT
20  PRINT "HELLO ";N$
30  PRINT
35  FOR J = 1 TO 2000
40  PRINT "HERE IS MY CATALOG "
45  D$ = CHR$ (4)
50  PRINT D$;"CATALOG"

```

```

10 REM *****
15 REM #
20 REM # BY DAVID R.FOOTE #
25 REM #
30 REM # AGE 10 #
35 REM #
40 REM *****
45 HOME
660 TEXT : HOME
670 S$ = " "
680 K$ = "!_$_!"
690 VTAB 10: HTAB 1
695 FOR I = 1 TO 115
700 PRINT S$ + K$
705 FOR T = 1 TO 20: NEXT T
710 HTAB 1
714 VTAB 5: PRINT "MONEY COMES AND MONEY GOES"
715 VTAB 10
720 S$ = S$ + " "
730 NEXT I
740 PRINT S$ + S$

```

```

10 REM *****
20 REM # BY DAVID R.FOOTE #
30 REM # ***** #
40 REM *****
45 HOME
50 PRINT "THIS PROGRAM WILL DISPLAY "
55 PRINT
60 PRINT "DIFFERENT COLORS ON THE SCREEN"
62 REM
65 FOR I = 1 TO 1900: NEXT I
80 GR :D = 0
86 Y = 1
90 HLIN 0,39 AT Y:Y = Y + 1: IF Y < 38 THEN GOTO 90
140 COLOR= INT (14 * RND (1)) + 1
150 D = D + 1: IF D < 15 THEN GOTO 86
330 TEXT : HOME
335 VTAB 10: HTAB 12: PRINT "THAT'S ALL FOLKS"
340 FOR T = 1 TO 2000: NEXT T
341 GR : COLOR= 6
343 HLIN 0,39 AT X
344 X = X + 1
345 IF X < 39 GOTO 343
360 COLOR= 11
370 PLOT 18,23
390 PLOT 19,23: PLOT 20,24: PLOT 21,25: PLOT 22,26: PLOT
22,26: PLOT 21,27: PLOT 20,28: PLOT 19,29: PLOT 18,29:
PLOT 17,28: PLOT 16,27: PLOT 5,26: PLOT 16,25: PLOT 17,24:
PLOT 18,23
400 COLOR= 1
410 PLOT 18,28: PLOT 19,28: PLOT 20,27: PLOT 17,27
420 COLOR= 11: PLOT 17,26: PLOT 20,26: COLOR= 14: PLOT 20,25:
PLOT 17,25
430 COLOR= 11: PLOT 18,27: PLOT 19,27
440 PLOT 19,26: PLOT 19,25: PLOT 19,24:
450 COLOR= 1: PLOT 18,26: COLOR= 11: PLOT 18,25: PLOT 18,24
460 COLOR= 15: PLOT 19,30: PLOT 19,31: PLOT 19,32: PLOT 19,33
470 PLOT 20,34: PLOT 21,35: PLOT 22,36: PLOT 23,37: PLOT
24,37:
480 PLOT 18,34: PLOT 17,35: PLOT 16,36: PLOT 15,37: PLOT 14,37
490 COLOR= 8: PLOT 7,38: PLOT 7,37: PLOT 7,36: PLOT 7,35: PLOT
7,34
500 VLIN 0,34 AT 7
505 B = 0
510 COLOR= 6: HLIN 0,39 AT B
520 B = B + 1
540 IF B < 20 THEN GOTO 510
550 COLOR= 12: PLOT 6,20: PLOT 8,20: PLOT 9,19: PLOT 9,18:
PLOT 9,17: PLOT 8,16: PLOT 7,16: PLOT 6,16: PLOT 5,17:
PLOT 5,18: PLOT 5,18
560 COLOR= 9: PLOT 7,19
565 FOR Z = 1 TO 4000: NEXT Z
570 TEXT : HOME
580 PRINT "HOW WAS THAT FOR A PICTURE?"

```

```

2  REM  BY CHRIS CLARK AGE 14
5  SPEED= 100
10 PRINT "HI, I'M APPLE II."
20 PRINT
30 PRINT "WHAT IS YOUR FIRST NAME?"
40 INPUT F$
50 PRINT
51 PRINT "AND YOUR LAST NAME?"
52 INPUT Q$
60 PRINT "WELL, ";F$;" ";A$;", I HAVE A PET BIRD."
65 PRINT
70 PRINT "HIS NAME IS:"
80 LET C$ = "BEEP"
90 PRINT C$
100 FOR V = 1 TO 1000: NEXT V
105 HOME
110 PRINT "IF YOU LISTEN, YOU CAN HEAR HIM....."
115 FOR G = 1 TO 1000: NEXT G
120 PRINT CHR$(7)
130 PRINT
140 PRINT "HE SAYS HE LIKES YOU, ";F$
150 FOR N = 1 TO 1000: NEXT N
155 SPEED= 255
160 PRINT "BYE, ";F$
170 REM GOTO 160

```

```

2  REM      (((((C A R S>>>)))
3  REM      BY JOHN AND MATT O'MALIA      AGES 12 AND 11
5  FOR A = 1 TO 28
10  HOME
20  PRINT
25  HTAB A
30  PRINT " XXXXXX"
33  SOUND = PEEK ( - 16336)
35  HTAB A
40  PRINT "XXXXXXXXXXXXX"
45  HTAB A
50  PRINT " 0          0"
55  PRINT "*****"
59  NEXT A
61  FOR D = 2 TO 19: HOME
62  HTAB 29: VTAB D
65  PRINT " XXXXXX"
67  HTAB 29
70  PRINT "XXXXXXXXXXXXX"
72  HTAB 29: VTAB D + 2
75  PRINT " 0          0"
80  HTAB 1: VTAB 5
85  PRINT "*****"
90  NEXT D

```

# GLOSSARY

## argument

The variable, number or string that appears in the parentheses of a function. Like:

INT(N)	has	N	as an argument
LEFT\$(W\$,3)	has	W\$ and 3	as arguments

## array

A set of variables that have the same name. The members of the array are numbered. The numbers appear in parentheses after the variable name. See dimension, subscript. Examples:

A(0)	is the first member of the array A
B\$(7)	is the eighth member of the array B\$
CD(3,I)	is a member of the array CD

## arrow keys

Two keys on the Apple computer that have arrows on them. They move the input cursor to the left and right.

## ASCII

Stands for American Standard Code For Information Interchange. Each character has an ASCII number. Some actions like line feed, carriage return and bell also have numbers.

## assertion

The name of a phrase that can be TRUE or FALSE. The phrase we called "something A" in an IF statement is an assertion. An assertion is also a numerical expression. See expression, TRUE, FALSE, logic, IF, something A. Example:

the assertion	"A" < > "B"	is TRUE
the assertion	3 = 4	is FALSE

## autostart ROM

A memory chip in later Apple II computers. It has a program that puts the machine into Applesoft BASIC as soon as the machine is turned on. See boot.

## beep

The sound an Apple makes. See peep.

## bell

The early teletype machines had a bell (like the bell on a typewriter). The Apple makes a "beep" sound instead.

## bells and whistles

A phrase going back to the early days of hobby computing. It means the personal computer was hooked up to do some interesting or spectacular things, like flash lights or play music.

## blank

The character that is a space.

## boot

Means to start up the computer from scratch. An easy thing to do with modern computers that have start up programs stored permanently in ROM memory. It was an involved procedure in the early days. Now it usually means to read in the disk operating system programs (DOS) from a disk.

**branch**

A point in a program where there is a choice of which statement to execute next. An IF statement is a branch. So is an ON . . . GOTO statement. A branch is not the same as a jump where there is no choice. See jump, IF.

**buffer**

A storage area in memory for temporary storage of information being inputted or outputted from the computer.

**call**

Using a GOSUB calls the subroutine. Putting a function in a statement calls the function. Call means the computer does what commands are in the subroutine or does the calculation that the function is for.

**carriage return**

On a typewriter, you push the lever that moves the carriage carrying the paper so a new line can begin. In computing, it means the cursor is moved to the start of the line, but not down to the next line. See line feed, CRLF.

**character**

Letters, digits, punctuation marks and the space are characters.

**clear**

Means erase. Used in “clear the screen” and “clear memory.”

**column**

Things arranged vertically. See row.

**command**

In BASIC a command makes the computer do some action, such as erase the screen and move the cursor to the upper left for the HOME command. See statement, expression. Some commands need expressions to be complete. Example:

HTAB 2+J=1

**concatenation**

Means sticking two strings together by using a “+” sign. For example:

"HI "+"THERE"      gives the string      "HI THERE"

**constant**

A number or string that does not change as the program runs. It is stored right in the program line, not in a box with a name on the front. See line.

**CRLF**

Short for “carriage return followed by line feed.” This is what is called just a “carriage return” on a typewriter. See carriage return, line feed.

**cursor**

A marker that shows where the next character on the screen or in a storage buffer will be placed. Cursor means “runner.” The cursor runs along the screen as you type. There are two kinds of cursors in the Apple:

INPUT cursor	a flashing square on the screen
PRINT cursor	invisible, “shows” where next character will be printed

**data**

BASIC has two kinds of data: numerical and string. Logical data (TRUE, FALSE) are types of numerical data.

**debug**

Means to run a program to find the errors and fix them. You fix the errors by editing the program. See edit.

**deferred execution**

Means run a stored program. See immediate execution.

**delay loop**

A part of the program that just uses up time and does nothing else. Example:

```
FOR T=1 TO 2000:NEXT T
```

**disk**

Short for "diskette." Someone called it "a cross between a 45 RPM record and a magnetic tape." Used to store information in a permanent form. Like a magnetic tape, the information can be erased and new information can be recorded.

**edit**

There are two kinds: editing a line and editing a program. In either kind, you are retyping parts of it to correct it.

**enter**

To put information into the computer by typing, then pushing the RETURN key. The information goes into the input buffer as it is typed. When RETURN is pushed, the computer uses the information.

**erase**

To destroy information in memory or write blanks to the screen. See clear.

**error trap**

Part of a program that checks for mistakes in information that the user has entered, or checks to see if computed results make sense.

**execute**

To run a program or to perform a single command or statement.

**expression**

A portion of a statement that has a single value, either a number or a string. See value. Examples:

```
7*X+1          3*LEN(D$)+RND(8)
GT$+LEFT$(D$,2) "DOPE "<> N$
```

**FALSE**

The number 0. See logic, assertion.

**file**

A collection of information that is on a disk or is in the computer ready to put on a disk.

**file name**

The name of a file. See lesson 14 for legal names.

**flashing box**

The input cursor.

**flashing mode**

The mode chosen by the command FLASH which makes all material that the program writes to the screen flash on and off. Text you type from the keyboard will not flash. See normal mode, inverse mode.

**fork in the road**

Describes a branch point in the program. See branch.

**function**

BASIC has a number of functions built in. Each function has a name followed by parentheses. In the parentheses are one or more arguments. The function has a single value (numerical or string) determined by its arguments. See value, argument. The functions treated in this book are:

ASC, CHR\$, INT, LEFT\$, LEN, MID\$, PDL,  
PEEK, RIGHT\$, RND, SCRN, STR\$, VAL

**graphics**

Means picture drawing.

**HELLO program**

Each disk that can be used to boot the computer has a special program written in BASIC that is executed automatically at the start. It is called the HELLO program.

**immediate execution**

When a line that does not start with a number is entered from the edit mode, it is executed as soon as the RETURN key is pressed. If the line has only one command, you usually think of it as "entering a command." But the line may have several statements separated by colons, and thus it is a little program. Then you think of it as "executing a program in the immediate mode."

**index**

An array name is followed by one or more numbers or numerical variables in parentheses. Each number is an index. Another word for index is "subscript."

**initialize a disk**

When you open a box of disks from the store, they are not yet ready to receive files. The computer must write some special things on them first. This is called "initialization of the disk."

**integers**

The whole numbers, positive, negative and zero.

**inverse mode**

The command INVERSE makes the computer write all output as black letters on white background. See normal mode.

**jump**

The GOTO command makes the computer jump to another line in the program, rather than execute the next line.



## line

There are two kinds of lines in BASIC: Lines that start with a number are stored in the program in memory. Lines that do not start with a number are executed right away (see immediate execution). A line contains one or more statements, separated by colons.

Parts of a line:

16	IF A<=7 THEN PRINT Q\$+LEFT\$("RAT",K):GOTO 40	
16		line number
IF A<=7 THEN PRINT Q\$+LEFT\$( , , , )		first statement
GOTO 40		second statement
IF A<=7 THEN		a command
PRINT Q\$+LEFT\$("RAT",K)		a command
GOTO 40		a command
A<=7		an assertion
A<=7		an expression
Q\$+LEFT\$("RAT",K)		an expression
LEFT\$("RAT",K)		a function
"RAT"		an argument
K		an argument
A, Q\$, K		variables
7, "RAT"		constants
<=		an operation

## line buffer

The storage space that receives the characters you type in. See buffer.

## line editing

Retyping parts of a line to correct it. You do this by moving the cursor to the wrong part and then typing the correct characters.

## line feed

Moving the cursor straight down to the next line. The ASCII number 10 signals this command to the screen or printer. See carriage return and CRLF.

## line numbers

The number at the beginning of a program line. The line number tells the computer where to store the line. Some lines don't have numbers (the ones that will be executed in the immediate execution mode).

## listing

A list of all the lines in a program.

## load

To transfer the information in a file on the disk to the memory of the computer by using the LOAD command.

## logic

The part of a program that compares numbers or strings. The relations AND, OR, and NOT, and =, <, >, <=, >, <=, and >= are used. See assertion, IF, something A.

**loop**

A part of the program that is done over and over again. There are two kinds of loops: FOR . . . NEXT loops, and “home made” loops that use IF . . . commands with GOTO commands.

**loop variable**

Is the number that changes as the loop is repeated. For example:

```
FOR I=1 TO 5:NEXT I      I is the loop variable
```

**memory**

The part of the computer where information is stored. Memory is made of semiconductor chips, but we think of it as “boxes” with a label on the front and the information inside.

**menu**

A list of choices shown on the screen. Each choice has a letter or number beside it. The program user presses a key to pick which choice is wanted.

**message**

The string in quotation marks after an INPUT command. Example:

```
INPUT"TEXT , NUMBER " ;T$,N
```

**monitor**

Has two meanings. We use it to mean a box with a TV type screen that is connected to the computer. It displays text and graphics but cannot receive television programs. In machine language programming, a monitor is a control program.

**nesting**

When one thing is inside of another. In a program we nest loops. Inside a statement, we can nest expressions or functions.

```
L=LEN(MID$(A$,3,J))      nested functions
X=5*(6+(7*(8+K)))        nested parentheses
```

**normal mode**

Is the white on black printing on a screen that the computer usually does. See flashing mode, inverse mode.

**number**

Is one type of information in BASIC. The numbers are generally decimal numbers. See integer, strings.

**operation**

In arithmetic: addition, subtraction, multiplication and division, with symbols + , - , \* , and /. The only operation for strings is concatenation.

**peep**

The sound that an Apple makes, usually to signal an error or some danger to the program. You can make the Apple peep with this command:

```
PRINT CHR$(7)
```

It is called “beep” in the Apple manuals and sometimes called “bell” because the old teletype machines used a bell for the same purpose.

**program**

There are two kinds. The usual program is a list of numbered lines containing statements. The computer executes the statements (commands) in order when the RUN command is entered. The program is stored in a special part of memory, and only one program can be stored at a time.

A one line program can be entered when the computer is in the edit mode. It does not start with a line number and runs as soon as you press the RETURN key. This one line program does not get mixed with the stored program. But when it runs, it may read or change the variables (if any) that the stored program made when it ran.

**prompt**

Is a little message you put on the screen with an INPUT to remind the user what kind of an answer you expect. Its name comes from the hint that actors in a play get from the prompter if they forget their lines.

**psuedo-random**

A number that is calculated in secret by the computer using the RND( ) function. It is usually called a "random number." Pseudo-random emphasizes that the number really is not random (since it is calculated by a known method) but just is not predictable by the computer user.

**punctuation**

The characters like period, comma, /, ?, !, \$, etc.

**random**

Numbers that cannot be predicted, like the numbers that show after the roll of dice, or the number of heads you get in tossing a coin 10 times.

**remark**

A comment you make in the program by putting it in a REM statement. Example:

```
REM THE NEXT 7 LINES ARE A SUBROUTINE
```

**reserved words**

A list of words and abbreviations that BASIC recognizes as commands or functions. The reserved words cannot be used in variable names. See appendix C.

**return a value**

When a function is used (called), its spot in the expression is replaced with a value (a number or a string). This is called "returning a value."

**RUN mode**

The action of the computer when it is executing a program is called "operating in the RUN mode." You get into the run mode from the edit mode by entering RUN. When the computer ends the program for any reason, it returns to the edit mode.

**row**

Things arranged horizontally (across).

**save to disk**

The program in the computer's memory is written (in magnetic code) on a diskette in the disk drive by the command:

## **SAVE filename**

where “filename” stands for a name you choose to identify the program while it is on the disk.

## **screen**

The TV screen or a similar one in a monitor that is hooked up to the computer. See monitor.

## **scrolling**

The usual way an Apple writes to the screen is to put the new line at the bottom of the screen and push all the old lines up. This is called “scrolling.”

## **something A**

Is a phrase in this book that stands for an assertion in an IF statement. See assertion, IF. Example:

```
IF A>4 THEN GOSUB 500      A>4 is “something A”
```

## **stack**

Is a data type used in machine language programming. The 6502 processor has a stack in page \$01 and it holds information about loops, subroutines, and nesting.

## **starting stuff**

Is the name given in this book to initialization material in a program. It includes REM's for describing the program, input of initial values of variables, set up of array dimensions, drawing screen graphics, and any other things that need to be done just once at the beginning of a program run.

## **statement**

The smallest complete section of a program. It starts with a command. The command may have expressions in it.

## **store**

To put information in memory or to save a file on a disk.

## **string**

A type of data in BASIC. It consists of a set of characters. See number.

## **subroutine**

A section of a program that starts with a line called from a GOSUB command and ends with a RETURN command. It may be called from more than one place in the program.

## **subscript**

A number in the parentheses of an array. It tells which member of the array is being used. See index.

## **syntax**

Means the way a statement in BASIC is spelled. SYNTAX ERROR means the spelling of a command or variable name is wrong, the punctuation is wrong or the order of parts in the line is wrong.

## **timing loop**

A loop that does nothing except use up time. See delay loop.

**title**

The name of a program or subroutine. Put it in a REM statement.

**TRUE**

Has the value 1. See logic, FALSE, assertion.

**typing**

Pressing keys on the Apple. It is different from “entering.” See enter.

**value**

The value of a variable is the number or string stored in the memory box belonging to the variable. See variable.

**variable**

A name given to a “box” in memory. The box holds a value. When the computer sees a variable name in an expression, it goes to the box and takes a copy of what is in the box back to the expression and puts it where the variable name was, and continues to evaluate the expression. See variable name.

**variable name**

A variable is either a string variable or a numerical variable. The name tells which. The rules for naming variables are given in lesson 20. The most important rule is that string variables have names ending in a “\$” sign. Numerical variables do not have a dollar sign on the end.

## INDEX OF COMMANDS AND FUNCTIONS EXPLAINED IN THIS BOOK

AND 149–150  
ASC( ) 138–140  
CATALOG 68, 70–72  
CHR\$( ) 138, 140–141  
COLOR 98, 100, 135  
CONT 162–165  
DELETE 68, 71–72  
DIM 145–146, 148  
END 112–113, 115, 163  
FOR. . . TO 57, 82–85  
FLASH 13, 16  
GET 108–11, 142–143, 160  
GOSUB 112–113  
GOTO 27, 38, 39, 40, 42–46, 73, 165  
GR 98, 100  
HLIN 98, 103–104, 159  
HOME 7–8  
HTAB 79–80, 92  
IF. . . THEN 27, 42–43, 58–60, 73, 76, 149  
INPUT 27, 28, 34, 49, 73–74, 108, 109, 143  
INIT 168  
INT( ) 63, 65, 67  
INVERSE 13, 16  
LEFT\$( ) 123–124  
LEN( ) 123, 125  
LET 34, 35, 48–53, 73–74  
LIST 17, 75, 121  
LOAD 68, 70, 72, 168, 170–171  
LOCK 168  
MID\$( ) 123, 126  
NEW 7, 9  
NEXT 57, 82–85  
NORMAL 13, 15  
NOT 149, 153  
ON. . . GOTO 143  
OR 149–150  
PDL( ) 132–133  
PEEK( ) 132–134, 143  
PLOT 98, 100–101  
PRINT 7, 9, 22, 27, 30–32, 49, 54, 73–74, 79, 164, 166  
REM 17, 18, 21, 22, 161  
RETURN 112–113  
RIGHT\$( ) 123, 125  
RND( ) 27, 63–65  
RUN 7, 9, 10, 87–90, 165  
SAVE 68–69, 72, 168–169, 171  
SCRN( ) 132–133, 136  
SPEED 30, 33  
STEP 84  
STOP 162–166  
STR\$( ) 48, 128–129  
TAB( ) 54–56, 79–80  
TEXT 98, 100  
UNLOCK 168  
VAL( ) 48, 108, 128–129  
VLIN 98, 103, 105, 159  
VTAB 79–80, 92

## KEYS EXPLAINED IN THIS BOOK

RESET 38, 39, 40, 70, 89, 132–135  
RETURN 10, 23, 108, 122, 139, 156  
REPT 23, 25, 122  
ARROW KEYS 24, 25  
SHIFT 10  
ESC 119–122  
CTRL 162, 164

## INDEX OF ERROR MESSAGES

### CAN'T CONTINUE

You used the CONT command when it was not allowed.

### DIVISION BY ZERO

You divided by zero. Or you divided by a variable whose value was zero.

### ILLEGAL DIRECT

You used INPUT or GET in the edit mode. (Or you used DEF FN or DATA in the edit mode.)

### ILLEGAL QUANTITY

You made one of these errors:

You used a negative number as an array subscript, like

LET A(-1)= 34

You used a function with the wrong kind of argument. Like:

wrong: L=VAL(R)	correct L=VAL(R\$)
wrong: TAB(-3)	correct TAB(3)
wrong: SPEED=400	correct SPEED=255
wrong: HTAB 55	correct HTAB 40

Wrong arguments may be a string where a number is needed or a number where a string is needed, or a negative number where a positive one is needed, or a number that is bigger than allowed.

### NEXT WITHOUT FOR

You used a NEXT before the computer reached a FOR . . . statement. Or you used the wrong name for the variable. Like:

```
FOR I= 1 TO 5
NEXT M
```

### OUT OF MEMORY

Usually it means you have nested too many FOR . . . NEXT loops or too many subroutines inside each other, or an expression has too many parentheses.

## **OVERFLOW**

You did a calculation which had a very large answer, too big for the computer to handle.

## **REDIM'D ARRAY**

You made one of these errors:

You used an array before you did the DIM command for it. Like:

```
LET A(3)=7:DIM A(20)
```

or you did executed DIM twice for the same array, like going through the DIM line twice.

```
2010 DIM B$(7)
```

## **RETURN WITHOUT GOSUB**

You let the computer reach a RETURN command before it went through a GOSUB . . . command. This usually happens when the program accidentally “runs into” a subroutine at the end.

## **STRING TOO LONG**

You used concatenation to make a string longer than 255 characters.

## **BAD SUBSCRIPT**

You made an error using an array. Like:

```
DIM A(5,5):A(1,1,1)=77      wrong number of subscripts
```

```
or DIM A(5): A(14)=7        subscript was larger than 5.
```

## **SYNTAX ERROR**

You “spelled” the line wrong. Maybe you forgot a ( or a ; or put a # in a name, etc.

## **TYPE MISMATCH**

You mixed numbers and strings, like:

```
LET A='9' or LET A$=33 or A$=LEFT(A$,1)
```

## **UNDEF'D STATEMENT**

You used a GOTO or a GOSUB to a line number that is not in your program.



## ALPHABETICAL INDEX

### A

addition 48, 63  
alphabetize 138  
annoyance detector 156  
Applesoft 70, 89  
argument 54, 56, 63, 123, 130  
arithmetic 48-52, 59, 63, 87  
array 145-148  
arrow 39, 83  
arrow keys 23-25, 119-122  
ASCII 138-141  
assertion 42-43  
autostart ROM 38

### B

bells and whistles 13  
blank space 15  
boxes, see memory boxes branch 27  
break 163-164  
buffer 87, 119

### C

calculator mode, see edit mode  
carriage return 138, 141  
character 15, 23, 30, 51, 54, 72, 87, 96, 108-109, 119, 124, 125, 138-143  
clear 17, 35, 43  
colon 17, 29, 54, 73, 75-77  
color 98-100, 106, 115, 132, 135-136  
column 55, 81  
comma 29-30, 72, 80  
command 8, 17, 19, 27, 58-59, 72, 80, 87, 113, 130  
command C 42-45, 59-60, 77, 152  
command mode, see edit mode  
concatenation 91, see gluing  
conditional test 42  
constant 15, 30, 51  
cursor 23-25, 30-31, 54, 56, 89, 108, 109, 119-122, 170

### D

debugging 87, 114, 156, 162-166  
decimal numbers 48-49, 63-65  
deferred execution mode, see run mode  
dice 66  
die, see dice dimension 145-148  
direct mode, see edit mode division 48-49, 135  
disk 68-71, 98, 167  
disk drive 68-69, 167  
dollar sign 28, 50, 96  
DO UNTIL 58 drawing 22, 26, 98, 107, 115, 132, 136

## **E**

edit a line 10, 23, 87  
edit mode 13, 30, 87-90, 113, 115, 152, 164  
enter a program 69, 90  
equal 48, 52-53, 59, 74, 154  
erase 8, 17-19, 31, 70, 91, 93, 115, 132, 135, 157, 159, 168  
error 90, 156  
error trap 160  
execute 88-89  
expression 17, 30, 42, 58, 67  
EXTRA IGNORED 29, 81

## **F**

false 42-45, 149-154  
file 69, 71-72, 171  
flashing cursor, see cursor, input cursor  
floating point 132  
flow of command 38, 42  
fork in the road 45  
format 156  
function 48, 56, 63-65, 67, 108, 123, 128-130

## **G**

Glossary 207  
gluing 37, 92, 109, 124-125  
graphics ... 54, 73, 79, 81, 98-100, 103-104, 132, 134  
greater than 59, 154

## **H**

HELLO program 7, 68, 167-168  
horizontal 103-105

## **I**

ILLEGAL QUANTITY ERROR 56, 80  
immediate mode, see edit mode  
index 145-148  
initialization 132  
initialize a disk 7, 68  
input 27, 75, 87, 109, 111  
input cursor 24, 30-31, 73  
instruction 54  
integer 63, 65

## **J**

jump 39-40

## **K**

keyboard 109, 138-143, 156  
keystroke 108

## **L**

less than 59, 154  
letters 15, 96, 109, 121  
life saver, 40, 70  
line numbers 12, 20, 38, 87, 112, 119, 121, 165  
line 32, 73, 77, 80, 82  
line, adding 20  
line feed 138, 141  
line, replace 17, 21  
line editing 17, 23, 24, 119-122  
list 18  
logic 42, 58, 73, 149  
loop 27, 38, 40, 54, 57-58, 82-85, 91, 106, 157-158, 166

## **M**

memory 11, 14, 17-19, 70, 88, 114, 121  
memory boxes 17, 19, 27-28, 34, 48, 50, 53, 91, 124, 126, 130, 143, 146, 151, 165  
menu 99  
message, in INPUT 27, 73, 109  
message, ERROR 29, 58, 72, 123, 166, 171  
minus sign 49  
modular 112  
monitor 98  
multiplication 48-49, 63, 65, 135

## **N**

name 28, 35, 72  
nesting 58, 63, 82, 84-85  
not equal 47, 154  
numbers 15, 27, 30, 48-49, 51-53, 56, 58, 63-65, 67, 72, 80, 83, 96, 109-111, 128-130, 138  
number, negative 63

## **O**

operation 48  
operating mode 89  
output 27  
output cursor 30, 79

## **P**

paddle 132-135  
Papert, Seymour iv  
parenthesis 56, 63-64, 67  
peep 13-14  
pixel 98  
PRINT, mixtures in 52  
PRINT cursor 30-32, 55  
program 11, 17-20, 27-28, 38, 54, 68, 73, 75-76, 87  
programs, fast 132, 143  
program, spaghetti 156  
programming, top down 112  
prompt 89, 108, 156-157, 159  
projectile 132  
punctuation 15, 72, 96, 139, 141

## **Q**

question mark 74-75

## R

random 27, 63-65  
remark 17, 21-22, 76  
replace 48, 53  
reserved words 95-96, 172  
run mode 87-90

## S

screen 8, 14, 17-18, 30-32, 35, 54, 56, 79, 108, 116, 132-136, 145, 156-157, 159  
scrolling 159  
semicolon 29-32, 54, 75  
sentence 109  
snipping strings 123-126  
something A 42-46, 59, 152  
space 30-32, 55, 72, 91, 94, 125  
speaker 138  
starting stuff 158  
statement 17, 19, 48, 52, 73-77, 82, 113, 130  
stop 38  
string 15, 27-30, 35, 37, 42, 49-52, 56, 59, 91, 111, 124, 128-130  
string constant 13, 15, 30, 48, 51  
string, empty 94  
string variable 27-28, 30, 50, 91, 96, 108  
structured programming 112  
subroutine 73, 82, 107, 112-118, 135-136, 157, 159  
subscript 146  
subtraction 48  
suit of cards 63  
SYNTAX ERROR 9, 55, 72

## T

tape cassette 68, 70, 72, 169-171  
target 132  
text 79, 98, 100  
true ... 43-46, 73, 76, 149-154  
truncating 63  
TYPE MISMATCH ERROR 52  
typing 8, 27, 109, 121

## U

user friendly 108, 156

## V

value 35, 48, 50, 54-55, 129-130, 165  
variables 17, 28, 30, 35, 50, 52, 67, 98, 101, 157  
variable, array 146  
variable, loop 54, 85  
variables, numerical 48, 50, 52, 96, 108, 146  
variable names .... 35, 48, 50, 81, 95-97  
vertical 103-105

## W

whole numbers, see integers  
word 109-110

## Z

zero 10



# KIDS AND THE APPLE

## A KIDS BOOK FOR ADULTS TOO? YOU BET!

The title of this book may be *KIDS AND THE APPLE*, but don't let it fool you. Designed for children ages 10 - 14, it also provides valuable information for the adult purchasing his or her first Apple II. *KIDS AND THE APPLE* was created to lead you gently, yet quickly, into the world of Applesoft Basic.

You'll learn how to write action games, board games and word games. Guidance, explanations, exercises, reviews and quizzes are given in an easy, non-technical style. Study guides precede each of the 33 chapters, and every new concept is tied to simple, everyday experiences. Also, as an added bonus, there are dozens of cartoons and illustrations that will have you laughing as you learn.

Computers are here to stay. *KIDS AND THE APPLE* prepares the computer generation by solving the mysteries of the Apple II in entertaining and enjoyable ways!

**ABOUT THE AUTHOR:** Edward H. Carlson has been a professor of Physics at Michigan State University since 1965. His interest in computers, which began in 1960, has taken many forms. He has been involved with numerous University projects and recently helped found a Computer Camp for children. Professor Carlson's other books include *KIDS & THE ATARI*, *KIDS & THE VIC*, *KIDS & THE TI 99/4A*, *KIDS & THE PANASONIC*, *KIDS & THE COMMODORE 64* and *KIDS & THE TIMEX/SINCLAIR*.

ISBN 0-88190-019-2



**DATAMOST**

8943 Fullbright Avenue, Chatsworth, CA 91311-2750  
(213) 709-1202

